Ontologiebasierte Modellierung perzeptiver Ressourcen an Bord unbemannter Luftfahrzeuge

M.Sc. Denis Smirnov

Vollständiger Abdruck der von der Fakultät für Luft- und Raumfahrttechnik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

1. Gutachter: Univ.-Prof. Dr.-Ing. Peter Stütz

2. Gutachter: Univ.-Prof. Dr.-Ing. Mark Minas

Die Dissertation wurde am 14.06.2018 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Luft- und Raumfahrttechnik am 04.10.2018 angenommen. Die mündliche Prüfung fand am 15.10.2018 statt.

Diese Arbeit ist als Dissertation an der Universität der Bundeswehr München in Neubiberg entstanden.

Bookcover graphic by Freepik.

Vorwort

„*Hell, It's About Time*“, um es mit den Worten von Tychus Findlay auszudrücken. 3.080 Dateien, 371.171 Zeilen Code und 53 296 Wörter später stehe ich vor der Vollendung meiner Dissertation, welche ich in meiner mehrjährigen Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Flugsysteme an der Fakultät für Luft- und Raumfahrttechnik der Universität der Bundeswehr München anfertigen durfte. Während der aufregenden und lehrreichen Zeit am Institut hatte ich die Gelegenheit, mit zahlreichen interessanten und großartigen Menschen zusammen zu arbeiten. Zu diesen zählen u.a. das Team um die Werkstatt, das Sekretariat und Kollegen anderer Universitäten. Ein Dankeschön an Euch alle.

Ein besonderes Dankeschön geht an Christian Hellert, mit dem ich zeitgleich meine Tätigkeit an der Universität begonnen habe. Das gemeinsame Engagement an dem Projekt *Sagitta*, aus dem wir sowohl wichtige Erkenntnisse für unsere Zukunft als auch Frustrationsmomente mitnehmen konnten, hat Christian zu einem geschätzten Kollegen gemacht, der mich immer mit Rat unterstützte und immer ein offenes Ohr hatte. Viel wertvoller jedoch als seine Unterstützung während der Promotion ist die auch durch private Interessen entstandene Freundschaft, welche ich sehr wertschätze und die weit über das Berufsleben reicht. Viel Erfolg bei Deiner eigenen Promotion und alles Gute für Deine Familie.

Ein großes Dankeschön geht weiterhin an die Kollegen rund um *Umbrella*. Die unzähligen gemeinsamen Stunden in den Laboren, allen voran der ganze Spaß, den wir hatten und die Momente, in denen man gemeinsam interessante Themen diskutierte, werden immer in Erinnerung bleiben.

Vielen Dank auch an Prof. Stütz, der es mir überhaupt ermöglicht hat, an einem so spannenden und interessanten Thema in solch einer aufregenden Umgebung zu arbeiten. Herzlich bedanken möchte ich mich auch bei Prof. Minas für das Interesse an meiner Arbeit und die Übernahme des Koreferats.

Abschließend möchte ich mich bei allen aktuellen und ehemaligen Kollegen des Instituts für Flugsysteme und des Projektes Sagitta bedanken. Durch Eure Kooperation und Fachgespräche, aber auch die Aktivitäten außerhalb des wissenschaftlichen Alltags, blicke ich auf eine Zeit zurück, die ich nicht missen möchte.

Ein besonderes Dankeschön geht an Johann Bergen, der trotz der vielen Änderungen sich immer wieder bereit erklärte, auch kurzfristig diese Arbeit Korrektur zu lesen.

0x49 0x6e 0x20 0x47 0x65 0x64 0x65 0x6e 0x6b 0x65 0x6e 0x20 0x61 0x6e 0x20 0x6d 0x65 0x69 0x6e 0x65 0x6e 0x20 0x56 0x61 0x74 0x65 0x72 0x2e

Denis Smirnov

Kurzfassung

Im Fokus dieser Arbeit steht ein Ressourcen- und Fähigkeitenmodell für unbemannte Luftfahrzeuge zur Durchführung von luftgestützten Aufklärungsmissionen. Dieses Modell soll Fähigkeits- und Zusammenhangswissen technischer Komponenten in geeigneter Weise aufbereiten, um ein maschinelles System in die Lage zu versetzen, zu entscheiden, ob, wann und wie solche Ressourcen im Kontext konkreter Aufklärungsaufgaben genutzt werden können. Als Ressourcen sollen von diesem Modell generell zur Verfügung stehende subsystemische Komponenten in Hardware (z.B. Sensoren, Rechner, Kamerahalterung) und Software (z.B. Sensorbildverarbeitung, Sensornachführung) erfasst werden.

Die wissenschaftliche Fragestellung zielt dabei auf die methodische Konzeptionierung, die prototypische Entwicklung und Untersuchung relevanter Eigenschaften des Modells vor einem Anwendungshintergrund ab. Letzterer ist beschrieben durch zwei Anwendungsfälle: zur Missionsvorbereitung und während der Missionsdurchführung. Im ersten Fall identifiziert das Modell, welche Konfigurationsvarianten, beschrieben durch Luftfahrzeugtyp und Payloadausstattung (z.B. Sensoren und Hardware), zur Verfügung standen, um eine spezifische Aufklärungsmission auszuführen. Zur Zeit der Missionsdurchführung wiederum wird das Modell an Bord eines spezifisch ausgerüsteten Luftfahrzeuges zur Bestimmung einzelner geeigneter Ressourcenkombinationen herangezogen.

Als Repräsentationsform zur Fähigkeiten- und Ressourcenmodellierung wurden Ontologiebeschreibungen in Form der Web Ontologie Language (OWL) genutzt. Das so implementierte Konzept wurde als Erweiterung in das am Institut für Flugsysteme an der Universität der Bundeswehr München entwickelte Sensor- und Perzeptionsmanagementsystem integriert. Die Validierung fand in drei Experimenten statt. Im Rahmen einer *strukturellen Bewertung* wurde das Modell, welches in einer Ontologie beschrieben ist, auf gängige Designpattern, allgemeine konzeptionelle Eigenschaften und domänenunabhängige Designentscheidungen hin untersucht. In einer *operationellen Bewertung* wurde die Systemfunktionalität bewertet. Hierfür wurde eine UAV Mission in einer simulierten Experimentalumgebung durchgeführt, bei der verschiedene perzeptive Aufgaben automatisiert ausgeführt werden sollten. Es wurde untersucht, wie sich das System in unterschiedlichen Situationen, abhängig von der Umgebung und verschiedenen Zuständen des Luftfahrzeugs, verhält und ob das entwickelte Konzept den im Rahmen der Arbeit definierten Anforderungen genügt. Die abschließende *Leistungsbewertung* evaluierte charakteristische Antwortzeiten des Modells, wobei zwei unterschiedliche Ansätze der Informationsabfrage miteinander verglichen wurden.

Inhaltsverzeichnis

1 EINLEITUNG 1

1.1 Unbemannte Aufklärungssysteme 3

1.2 Zukünftige operationelle Anforderungen und Automatisierungsansätze 5

1.3 Problemstellung 6

1.4 Aufgabenstellung 10
1.4.1 Funktionale Vorgaben 10
1.4.2 Methodische Vorgaben 11
1.4.3 Zusammenfassung 11

1.5 Wissenschaftlich-technischer Beitrag 11

1.6 Gliederung der Arbeit 12

2 GRUNDLAGEN 13

2.1 Ablaufbeschreibung einer Aufklärungsmission 13

2.2 Automatisierte Objekterkennung in Bilddaten 15
2.2.1 Verarbeitungskette 15
2.2.2 Beispiel einer variierenden Kette 17

2.3 UAV-bezogene Ressourcen und Fähigkeiten 18
2.3.1 Ressourcen 18
2.3.2 Fähigkeiten 22

3 ANFORDERUNGEN AN DAS MODELL 25

3.1 Integrierbarkeit in das bestehende Systemkonzept 25

3.2 Aussagefähigkeit hinsichtlich möglicher Ressourcenkombinationen 26

3.3 Reaktionsfähigkeit bei Zustandsveränderungen 27

3.4 Einhaltung begrenzter Verarbeitungszeiten 27

3.5 Domänenübergreifende Konzeptrepräsentation 28

4 ERWEITERUNG DES S&PM 31

4.1 Technische und funktionale Randbedingungen 31
4.1.1 Bewertung auf Modulebene 31
4.1.2 Ressourcenerweiterung im Rahmen von Aufklärungsmissionen 32
4.1.3 Integration des Ressourcenmodells 32

4.1.4 Kommunikation und Wartung ... 33
4.1.5 Kompatibilität zwischen Realflug und Simulation ... 34

4.2 Aufbau und Funktionsweise des erweiterten S&PM ... 34

4.3 Perzeptionsmodule, -Ketten und –Graphen ... 38
4.3.1 Perzeptionsmodule ... 38
4.3.2 Perzeptionskette ... 40
4.3.3 Perzeptionsgraph ... 41

4.4 Fähigkeiten und Ressourcen-Out/Inputs ... 41
4.4.1 Die Beschreibung $\boldsymbol{RCapability}$ ... 43

5 REPRÄSENTATIONSFORMEN FÜR DAS RESSOURCENMODELL ... 47

5.1 Formalismen zur Wissensrepräsentation ... 50
5.1.1 Aussagenlogik ... 51
5.1.2 Prädikatenlogik ... 53
5.1.3 Semantische Netze ... 55
5.1.4 Frames ... 57
5.1.5 Beschreibungslogiken ... 59
5.1.6 Ontologien ... 63

5.2 Formalismenbewertung- und Auswahl ... 64

5.3 Umsetzung mittels einer Ontologie ... 66
5.3.1 Ontologietypen ... 70
5.3.2 Beschreibungssprachen ... 72
5.3.3 Ontologiebeispiele zur Ressourcenmodellierung ... 86

6 IMPLEMENTIERUNG ... 93

6.1 Modulkommunikation und -synchronisation ... 93

6.2 Ontologieumsetzung und -integration ... 95
6.2.1 Ontologieintegration Dritter ... 99
6.2.2 Klassenbeschreibung ... 104
6.2.3 Data- und Objectproperties ... 107
6.2.4 Modulkonkatenation ... 111
6.2.5 Ontologieinstanzen ... 113

6.3 Ontologiedatenbank ... 124

6.4 Umsetzung des Perception Solvers ... 127

6.5 Umsetzung des Resource Managers ... 130
6.5.1 Resource Manager Proxy (C++) ... 130
6.5.2 Resource Manager (Java) ... 132

6.6 Umsetzung des Processing Coordinator ... 134
6.6.1 Die Modulverwaltung ... 134

6.6.2 Module und ihre Ausführung 138

7 EVALUIERUNG 141

7.1 Strukturelle Bewertung 142
7.1.1 Untersuchungsgegenstand 142
7.1.2 Methode und Experimente 143
7.1.3 Ergebnisse 151
7.1.4 Fazit der strukturellen Bewertung 157

7.2 Operationelle Bewertung 158
7.2.1 Missionsvorbereitung 159
7.2.2 Missionsdurchführung 166
7.2.3 Fazit der operationellen Bewertung 178

7.3 Leistungsbewertung 178
7.3.1 Untersuchungsgegenstand 179
7.3.2 Methode und Experimente 179
7.3.3 Ergebnisse 181
7.3.4 Fazit der Leistungsbewertung 184

7.4 Bewertung 185
7.4.1 Anforderungen an das Modell 185
7.4.2 Grenzen des Modells 186

AUSBLICK 189

QUELLENVERZEICHNIS 191

ANHANG 201

A. Datenbank Abfragen für 7.3.2.2 201

B. Datenbank Abfragen für 7.3.2.3 202

1

Einleitung

Der Einsatz von Luftfahrzeugen zur sensorgestützten Gewinnung geographischer und situativer Informationen auf der Erdoberfläche weist ein vielfältiges Aufgabenspektrum auf. Zu den zivilen Aufgaben gehören z. B. der Schutz von Infrastruktur, Menschenleben und Grenzen, Überwachung von Industrieanlagen [1] [2], Katastrophenschutz [3], die Ertragssteigerung und Kostenminderung in der Agrarwirtschaft (*Precision Farming*) [4] [5] [6] sowie das Gewinnen von Daten im Bereich der Kartografie („Fernerkundung"). Im militärischen Bereich sollen entsprechende Informationen („Luftaufklärung") den eigenen Truppen einen taktischen Vorteil verschaffen bzw. dem Schutz dieser dienen. Abbildung 1 zeigt Beispiele solcher Sensorinformationen.

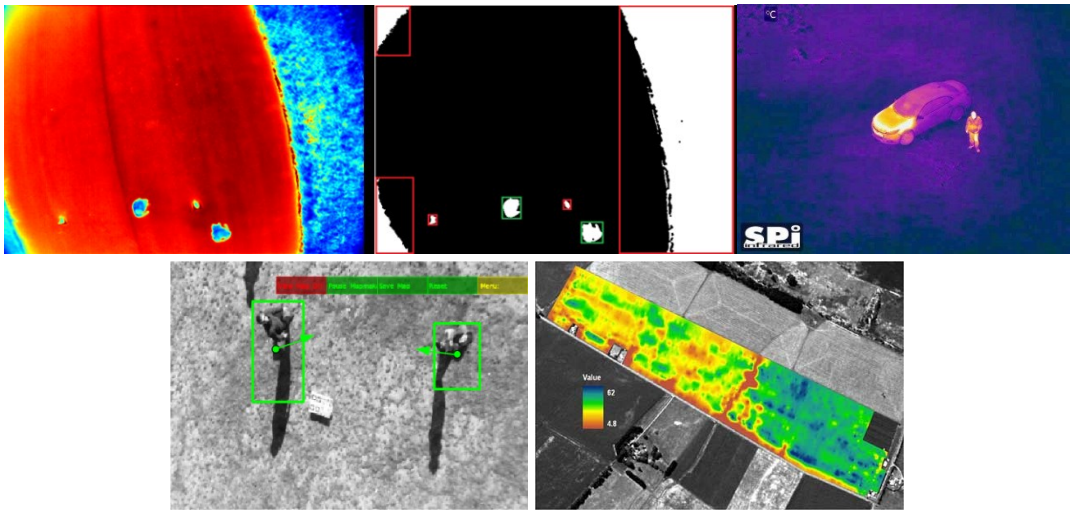


Abbildung 1: Beispiele für Sensordatenauswertung aus der Luft. Von links nach rechts: Farbkodiertes Infrarotbild, binäres Infrarotbild mit annotierten Personen [7]**, RGB-kodiertes Infrarotbild eines Fahrzeugs und einer Person[1], elektrooptisches Bild mit annotierten Personen** [8]**, bildgestützte Ertragsdichte eines Weizenfeldes[2].**

[1] Bildquelle: https://tinyurl.com/ybf4nekc, 16.04.2018
[2] Bildquelle: https://tinyurl.com/y9dsobdy, 16.04.2018

Um den unterschiedlichen Anwendungsfeldern gerecht zu werden, wurden zunächst vorhandene bemannte Luftfahrzeuge umgerüstet oder speziell für diese Zwecke entwickelt und mit entsprechender Sensorik sowie, soweit notwendig, dedizierten Arbeitsplätzen zur Bedienung ausgestattet. Beispiele aus zivilen Anwendungen hierzu sind etwa das Aufklärungsflugzeug DO 228 LM der Marine, mit dem z.B. Ölverschmutzungen detektiert werden, oder die DA 42 MPP (*Multi Purpose Platform*) Guardian der Firma Diamond Aircraft, ausgestattet mit einem beweglichen Sensorkopf. Im Bereich der öffentlichen Sicherheit kommen insbesondere Drehflügler zum Einsatz, um etwa Vermisste aufzuspüren oder den Straßenverkehr zu überwachen (vgl. Abbildung 2).

Abbildung 2: Zivile Luftfahrzeuge zur Fernerkundung[3]

Die unterschiedlichen Ausführungen, insbesondere von Luftfahrzeugen für militärische Aufklärungsmissionen, reflektieren ebenfalls die verschiedenen Anwendungsgebiete. Für die strategische, weitreichende Aufklärung wurden bis in die neunziger Jahre hochfliegende Höhenaufklärer, wie etwa die U-2 (vgl. Abbildung 3) eingesetzt. Für die gefechtsfeldnahe taktische Aufklärung in geringen und mittleren Höhen fiel die Wahl häufig auf schnellfliegende Kampfflugzeuge (vgl. Abbildung 3).

Abbildung 3: Militärische Luftaufklärungsflugzeuge. Links: U-2[4]. Rechts: Ein Tornado[5] mit einem Recce-Pod an der Unterseite.

[3] Bildquellen: https://tinyurl.com/y9kkltez, https://tinyurl.com/ybnt9fft, https://tinyurl.com/y9srx3u2, https://tinyurl.com/yazcg4kj, 16.04.2018

[4] Bildquelle: https://tinyurl.com/ybmk4v83, 16.04.2018

[5] Bildquelle: https://tinyurl.com/ya5bs9bp, 16.04.2018

1.1 Unbemannte Aufklärungssysteme

In den letzten Jahren haben sich unbemannte Luftfahrzeuge (engl.: *Unmanned Aerial Vehicle* – UAV) [9],[10] zunächst im militärischen Bereich, aber zunehmend auch für zivile Anwendungen als sensortragende Luftfahrzeuge für die genannten Anwendungsfelder durchgesetzt (vgl. Abbildung 4).

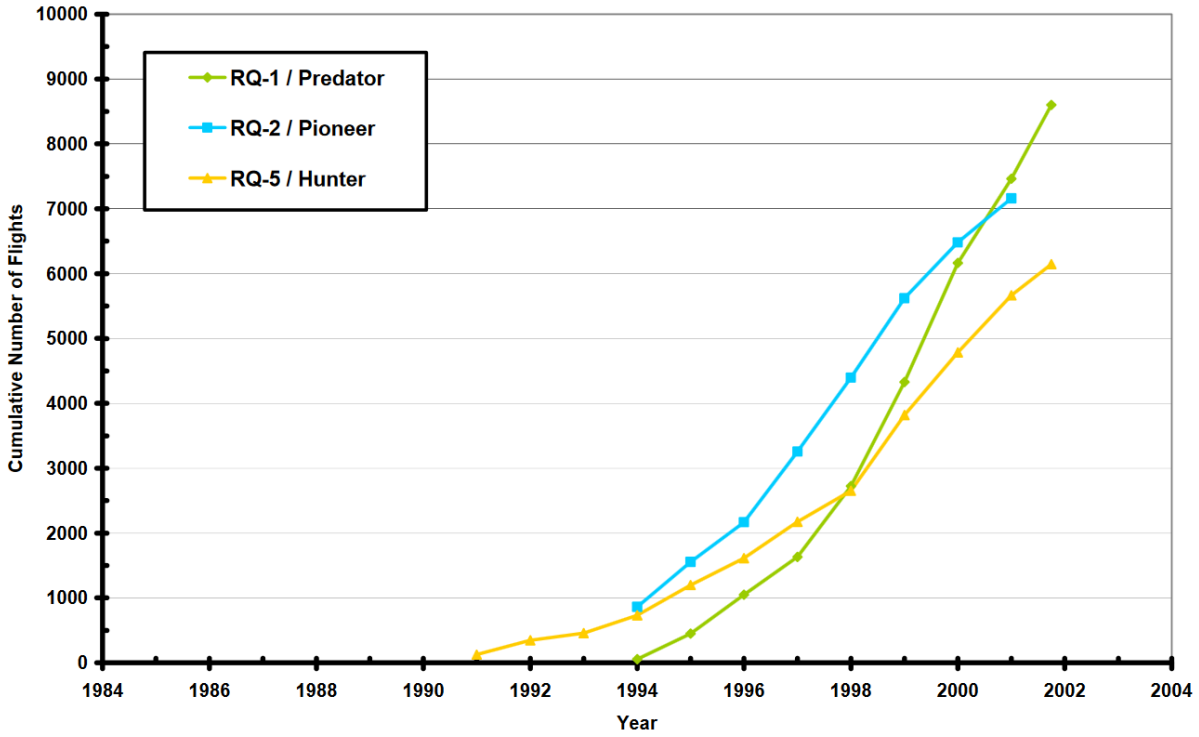


Abbildung 4: Anzahl der Flüge des US Militärs mit der Predator, Pioneer und Hunter Drohnen zwischen den Jahren 1984 und 2004 [11]

Die Vorteile gestalten sich vielfältig: Zunächst ergeben sich geringere Betriebskosten [12], da die Luftfahrzeuge mit vergleichbarer Aufklärungsnutzlast durch Verzicht auf Cockpit und Lebenserhaltungssysteme einfacher und leichter gebaut werden können. Zudem sinken die Kosten für Wartung der Systeme und Ausbildung der Piloten. Darüber hinaus können diese nun für Missionszeiten und -profile ausgelegt werden (bis zu 48 Std mit konventionellem Antrieb bzw. prinzipiell unbegrenzt durch Solarantrieb (vgl. Abbildung 5), Flughöhen bis zu FL 700[6]), welche mit bemannten Luftfahrzeugen nur mit größtem Aufwand zu realisieren sind.

[6] FL bedeutet *Flight Level.* Die Zahl dahinter steht für ein Hundertstel der Höhe in Fuß, die dem augenblicklich vom Höhenmesser gemessenen Luftdruck über der Standardatmosphäre von 1013,25 hPa entspricht.

Abbildung 5: Beispiele für UAVs. Links die MQ-9 Reaper[7] mit einer Flugdauer von über 48 Stunden. Rechts das von Airbus entwickelte HAPS UAV Zephyr[8] mit einer absolvierten Flugdauer von mehr als 10 Tagen.

Weiterhin können Flugmanöver durchgeführt werden, die für einen Piloten an Bord eine zu große körperliche Belastung darstellen würden. Aus militärischer Sicht ist darüber hinaus die Gefahrenvermeidung für die eigene Besatzung ein ausschlaggebendes Argument. Ein weiterer Vorteil ist die Möglichkeit, den Einsatzort der UAVs und den Ort der Besatzung geographisch zu trennen.

Die Luftfahrzeugtypen reichen mittlerweile von sog. „Micro-UAVs“ bzw. „Nano-UAVs“ (z.B. PD-100, ~28 g, ~12 cm Rotordurchmesser) bis hin zu großen Höhenplattformen mit bis zu 1,4 Tonnen maximalem Abfluggewicht (RQ-4B Global Hawk) und einer Spannweite von ~40 Meter (RQ-4E Euro Hawk). Sowohl im militärischen als auch im zivilen Sektor steigt die Vielfalt in Größe, Form, Ausstattung und Leistung von UAVs [13], [14]. Abbildung 6 zeigt zwei unterschiedliche UAV-Systeme: Links im Bild das „*Pocket-UAV*“ PD-100 für sogenannte „*Over the Hill*“ Aufklärungsaufgaben von Soldaten. Rechts im Bild das HALE (*High Altitude Long Endurance*) UAV Euro Hawk der Firma Northrop Grumman.

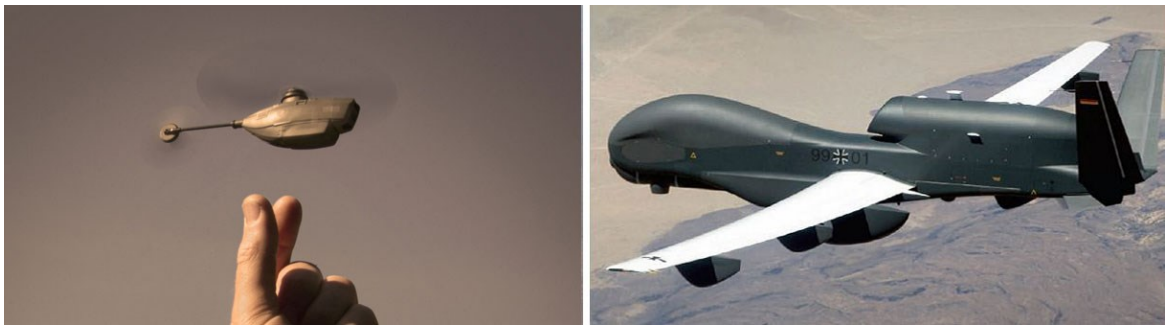


Abbildung 6: Unterschiedliche UAV Systeme, basierend auf Größe und Abfluggewicht. Links das PD-100 Nano UAV der Firma Prox Dynamics[9]. Rechts das RQ-4E Euro Hawk[10] UAV der Firma Northrop Grumman.

Betrachtet man ein sogenanntes *Unmanned Aerial System* (UAS), so lassen sich zunächst drei Hauptkomponenten unterscheiden. Das eigentliche Luftfahrzeug (*Unmanned Aerial Vehicle*,

[7] Bildquelle: https://tinyurl.com/y8qzkxrw, 16.04.2018
[8] Bildquelle: https://tinyurl.com/yb4czrvs, 16.04,2018
[9] Bildquelle: https://tinyurl.com/ptfq2m3, 16.04.2018
[10] Bildquelle: https://tinyurl.com/y95qkkmt, 16.04.2018

UAV), die Bodenkontrollstation (*Ground Control Station*, GCS), sowie den verbindenden Datenlink, welcher der Übermittlung von Führungs- und Bedienkommandos zur Plattform bzw. dem Senden von Flugdaten- und Aufklärungsinformationen zur GCS dient. Beim Einsatz von Satellitenkommunikation kann die Distanz zwischen GCS und UAV globale Dimensionen annehmen.

Die Führung von taktischen und größeren UAS wird konventionell (vgl. Abbildung 7) durch ein mehrköpfiges Team vorgenommen, welches sich aus UAV-Führungsoperateur (Pilot) und Sensoroperateur zusammensetzt [15]–[17].

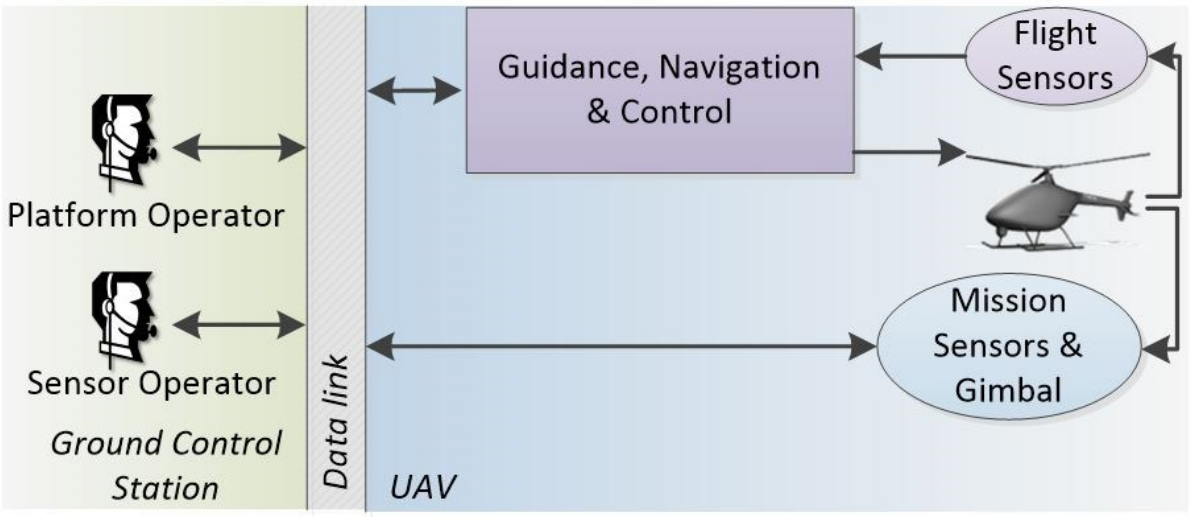


Abbildung 7: Konventionelles Systemkonzept zur Missionsführung eines UAVs

1.2 Zukünftige operationelle Anforderungen und Automatisierungsansätze

Um weiter Personalkosten zu senken und die Missionseffizienz zu steigern, befassen sich zur Zeit Forschungsprojekte [18] mit dem Ziel, die Anzahl der notwendigen Besatzungsmitglieder zu reduzieren bzw. die Anwendungsfelder auszuweiten. In diesem Kontext sollen drei Stoßrichtungen erwähnt werden:

- ***Single Operator-Single UAV***: Dieses Konzept sieht vor, dass ein Operateur vom Boden aus sowohl die Aufgabe des Piloten als auch die des Sensoroperateurs übernimmt. Durch Automatisierungsansätze in der UAV-Navigation und Sensordatenverarbeitung soll die Arbeitsbelastung des Operateurs soweit reduziert werden, dass die einzelne Person in der Lage ist, sowohl die UAV-Plattform zu bedienen als auch das Sensorsystem einzusetzen und Sensordaten auszuwerten.
- ***Single Operator-Multiple UAV***: Hierbei ist ein Operateur am Boden für mehrere UAVs verantwortlich. Dies erfordert letztlich den Einsatz veränderter Führungsparadigmen (Bsp.: *Task-Based Guidance* [19]), welche es dem Operateur ermöglichen, seinen Bedien- und Überwachungsaufgaben parallel für die unterschiedlichen UAVs nachzukom-

men. Einen weiteren konzeptuellen Beitrag in diese Richtung stellt das sogenannte *Tactical Control Systems* [20] dar. Dieses soll die Bedienkonzepte der unterschiedlichen Plattformen standardisieren und Zugriff darauf erlauben.

- ***Manned-Unmanned Teaming*:** Das anspruchsvollste Konzept sieht die Führung von mehreren UAVs von einem bemannten Luftfahrzeug aus vor. Hierbei kommandiert ein Besatzungsmitglied, welches bereits Aufgaben im Kontext der Führung des eigenen Luftfahrzeuges (Flugzeug, Hubschrauber) wahrzunehmen hat, eine Flotte von UAVs. Damit diese Konfiguration möglich ist, ist es notwendig, dass die UAVs in weit gesetzten Grenzen selbständig agieren, d.h. in der Lage sind, aufgabenorientierte Anweisungen des Piloten zu interpretieren, sich gegebenenfalls zu koordinieren und basierend darauf ihr Navigationsverhalten zu planen und sensorbezogene Aufgaben selbstständig durchzuführen [21].

Vor dem Hintergrund dieser zukünftigen operationellen Anforderungen für unbemannte Aufklärungssysteme stellt sich die Frage, wie durch geeignete Systemautomatisierung einer potentiellen Überlastung des Operateurs vorgebeugt werden kann oder, z.B. im Falle eines Datenlinkverlustes, die Aufklärungsmission vom technischen System autonom bis zur Wiederherstellung der Datenverbindung durchgeführt werden kann. Eine solche Erhöhung des Automatisierungsgrades betrifft dabei zwangsläufig sowohl die Flugführung des UAVs als auch die Bedienung der Missions-Sensorik und die Auswertung des Datenmaterials.

Bisherige Lösungsansätze in diesem Bereich haben ihren Automatisierungsschwerpunkt vor allem in der automatisierten Missionsplanung sowie der Missionsführung des UAVs ([22] [23]), während der eigentliche Sensoreinsatz im Bereich der Sensordatenakquirierung und des Sensordatenmanagements, der Sensorsteuerung und Sensordatenauswertung nur sehr vereinfacht betrachtet wurde.

Die vorliegende Arbeit entstand im Rahmen von Aktivitäten, die zum Ziel haben, den letzteren Aspekt zu detaillieren und hierfür geeignete Automatisierungsansätze zu finden.

1.3 Problemstellung

Die am häufigsten für Aufklärungszwecke eingesetzten Sensoren basieren auf elektrooptischen Prinzipien, sowohl im sichtbaren als auch im Infrarot Wellenlängenbereich (NIR-nahes Infrarot, MIR-mittleres Infrarot, FIR-fernes Infrarot). Weiterhin werden RADAR- und laserbasierte Sensoren für anderweitige Anwendungen eingesetzt. Diese ermöglichen es, dreidimensionale Bilddaten (z.B. Höhenprofil) von Objekten oder Landschaften zu erzeugen. Neben den Sensoren zur Bilddatenerzeugung werden an Bord eines UAVs weiterhin Hard- und Softwarekomponenten zur Verarbeitung und Übertragung der Daten benötigt.

Neben den internen Abhängigkeiten der sich an Bord befindlichen Komponenten, spielen externe Faktoren, wie z.B. Wetterlage und Tageszeit, eine entscheidende Rolle, wenn es darum geht, konkrete Aufklärungsinformationen im Rahmen einer Flugmission zu gewinnen. So liefert z.B. ein elektrooptischer Sensor ohne Tageslicht keine verwertbaren Informationen.

Durch das wachsende Aufgabenspektrum, welches durch den Einsatz von unbemannten Luftfahrzeugen abgedeckt werden soll, zeigt sich auch, dass das (erfolgreiche) Durchführen einer Mission nicht nur von der Wahl eines geeigneten UAVs und dessen Ausstattung abhängig ist, sondern dass vor allem auch der Operateur eine wesentliche Komponente in der Leistungskette darstellt. Wie bereits im vorhergehenden Kapitel 1.2 beschrieben, steigen die Anforderungen an den Piloten bzw. an den Operateur. Damit steigt ebenfalls die Gefahr der Überlastung. Dieser gilt es entgegenzuwirken. Dies gilt sowohl für das Führen des UAVs, als auch für die Konfiguration und die Benutzung der Sensorsystems sowie das Auswerten der Bilddaten. Auch hier (besonders bei schlechter Qualität der Bildinformationen) kann eine Überbeanspruchung des Operateurs zu Missionsfehlern führen [24] [25].

Es ist anzunehmen, dass die in Kapitel 1.2 bereits erwähnten zukünftigen Anforderungen dieses Problemfeld weiter verschärfen. Kontrolliert ein Operateur vom Boden bzw. ein Pilot aus der Luft mehrere UAVs, nimmt das Überwachen aller relevanten Datenströme einen (zu) großen Teil der Aufmerksamkeit des Operateurs bzw. Piloten ein. Untersuchungen dieser Problematik wurden u.a. in [26] und [27] durchgeführt. Während in diesem Beispiel die Aufmerksamkeit aufgrund einer zu hohen Datenflut leidet, sinkt die Aufmerksamkeit aufgrund von Unterforderung durch monotone Aufgaben, wie z.B. bei sogenannten „*Change Detection*“ bzw. „*Pattern of Life*“ Missionen [28]. Eine solche Luftaufklärungsmission, in der ein Gebiet kontinuierlich über mehrere Stunden überwacht werden muss, bietet nur wenig Abwechslung. Dadurch sinkt die Konzentration des Operateurs, und wichtige Veränderungen werden eventuell zu spät oder gar nicht bemerkt.

Um geeignete Automatisierungsansätze in diesem Arbeitsumfeld zu finden, kann zunächst das Tätigkeitsspektrum eines Sensoroperateurs analysiert werden. Dabei lassen sich folgende Teilaufgaben identifizieren:

1. Analyse der Aufklärungsaufgabe
2. Auswahl, Parametrisierung und Führung des geeigneten Sensors gemäß Aufgabe und Umweltsituation
3. (Erst-)Auswertung der Sensordaten
4. Kommunikation mit UAV-Pilot

Um die Belastung des Operateurs zu reduzieren, kann zunächst aus 3. auf den Einsatz maschineller Wahrnehmungsfunktionen geschlossen werden. Automatisierte Systeme aus dem Bereich der Bildverarbeitung und Mustererkennung [29] - [33] an Bord des unbemannten Luftfahrzeugs unterstützen bereits bei der Auswertung der Sensordaten. Bildverarbeitungsalgorithmen können automatisiert Veränderungen im Bild erkennen, gesuchte Objekte detektieren bzw. annotieren und den Operateur auf die Veränderung bzw. Erkennung aufmerksam machen.

Um auch die weiteren oben gelisteten Arbeitsschritte maschinell begleiten bzw. ersetzen zu können, stellt sich die Frage, wie ein assistierendes maschinelles System, analog zum menschlichen Sensorbediener, die insgesamt zur Verfügung stehenden Systemkomponenten intelligent einsetzen kann, um eine bestimmte Aufklärungsaufgabe zu erfüllen.

In der Literatur wird daraus folgend die Notwendigkeit für ein *Sensor Management System* diskutiert. Musick und Malhorta fassen das Ziel eines solchen Systems wie folgt zusammen:

> *„The goal of the SMgr (Sensor Manager) is to select the right sensor to perform the right service on the right object at the right time."* [34]

Die Schwerpunkte von Management Systemen schwanken stark, abhängig von dem Ziel bzw. der Aufgabe, die das System erfüllen muss. Manche Arbeiten konzentrieren sich z.B. ausschließlich auf einen Sensortyp, wie z.B. Radar [35], andere legen Wert auf die Ablaufkoordination (*Scheduling*), die Sensoreinsatzplanung und die Priorisierung der Sensoren [36]. Wiederum andere setzen den Schwerpunkt auf die Interaktion zwischen Sensorbewegung und die Flugführung mit dem Ziel, den Sensor möglichst optimal zu führen, um eine hohe Sensorabdeckung des Missionsgebietes zu erreichen [37],[38].

In diesem Zusammenhang wurde von Russ et. al [39] das Paradigma des *Sensor- & Perzeptionsmanagements* (S&PM) eingeführt und durch [42],[44] erweitert. Eine konzeptionelle Fassung davon ist Abbildung 8 zu entnehmen.

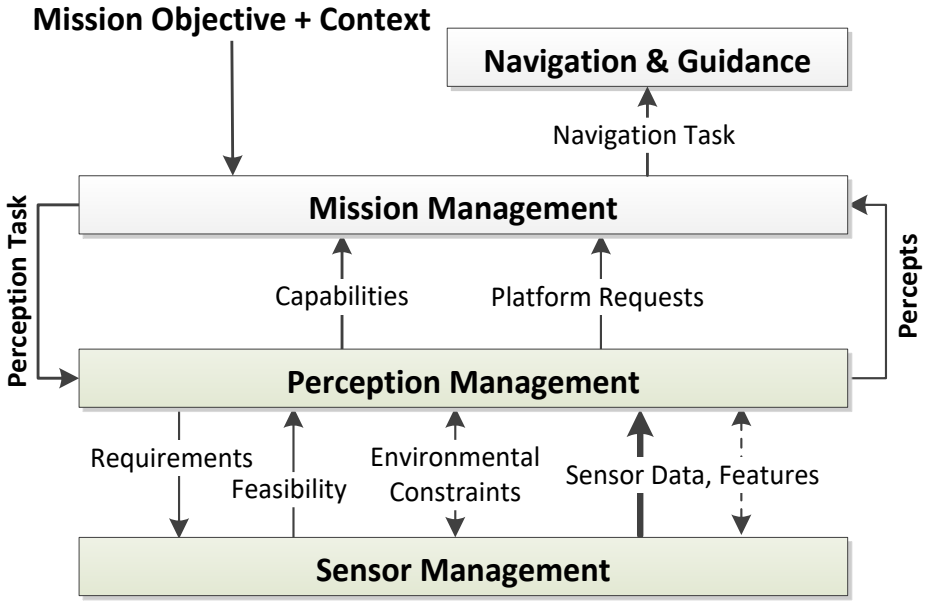


Abbildung 8: Übergeordnetes Systemkonzept des Sensor-& Perzeptionsmanagement

Das Systemkonzept besteht aus einer *Perception Management* Schicht und einer darunterliegenden *Sensor Management* Schicht. Das *Mission Management* stellt dabei ein übergeordnetes System dar, welches mit dem S&PM in Verbindung steht. Zu Beginn übermittelt das *Perception Management* alle vorhandenen Aufklärungsfähigkeiten (*Capabilities*) an das *Mission Management*. Basierend darauf ist das *Mission Management* in der Lage, spezifische Perzeptionsaufgaben (*Perception Task*) an das *Perception Management* zu delegieren. Dieses prüft die Durchführbarkeit der Aufgabe, indem es Anforderungen (*Requirements*), Machbarkeit (*Feasibility*) und Umgebungsbeschränkungen (*Environmental Constraints*) mit dem *Sensor Management*

abstimmt. Kann die Aufgabe durchgeführt werden, werden die notwendigen Prozeduren gestartet und die Aufgabe ausgeführt. Nach Abschluss werden die Ergebnisse (*Percepts*) zurück an das *Mission Management* geschickt. Neben den Fähigkeiten kann das *Perception Management* weiterhin *Platform Requests* - Luftfahrzeug betreffende Änderungswünsche wie z.B. Flughöhe oder Geschwindigkeit - an das *Mission Management* senden. Das Durchführen dieser Änderungswünsche obliegt dem *Mission Management*. Diese *Requests* sind notwendig, falls eine Perzeptionsaufgabe zwar theoretisch durchgeführt werden kann, jedoch aufgrund bestimmter Beschränkungen (wie z.B. eine zu große Flughöhe) erst dann durchgeführt wird, wenn diese Beschränkung aufgehoben ist. Weitere Details und eine Erweiterung des Paradigmas werden in Kapitel 4 diskutiert.

Das S&PM wird, wie in Abbildung 9 abgebildet, in die Gesamtsystemkonfiguration integriert. Aus den in Kapitel 1.2 vorgestellten Automatisierungsansätzen soll ein Operateur nun in der Lage sein, sowohl das Luftfahrzeug als auch die Missions-Sensorik (vgl. Abbildung 7) zu bedienen.

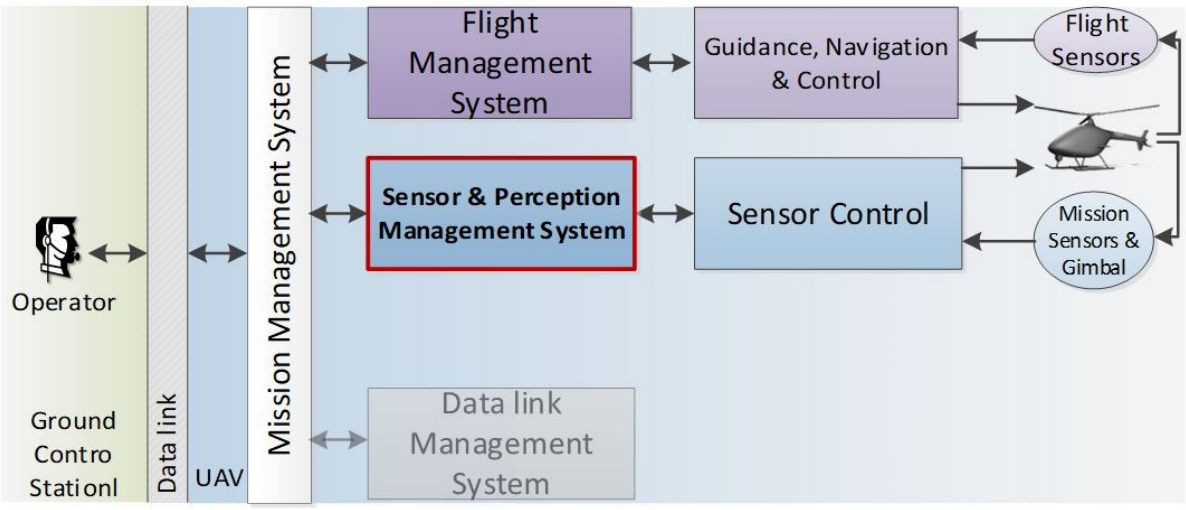


Abbildung 9: Systemkonfiguration einschließlich Subsysteme an Bord des UAVs bei einer Single Operator-Single UAV-Konfiguration

Das *Mission Management System* dient als Kommunikationsschnittstelle mit den Subsystemen des unbemannten Luftfahrzeugs und des Operateurs. Das *Flight Management System* ist für die navigatorische Führung des UAVs verantwortlich. Es kann auf verschiedene Arten (z.B. via Wegpunkten oder einer aufgabenorientierten Navigation, bei der das System an Bord des Luftfahrzeugs die Route selbstständig plant) das UAV zum Ziel führen. Das *Sensor & Perzeption Management System* ist verantwortlich für die Kontrolle der Sensorik (Sensorwahl, Parametrisierung, Ausrichtung etc.).

Um, wie beschrieben, Aufklärungsaufgaben automatisiert durchführen zu können, ist es grundlegend, analog zum menschlichen Operateur Zusammenhangswissen über die vorhandenen maschinellen Aufklärungsfähigkeiten zu verwenden und dieses mit den von extern zugeführten

Aufklärungsaufgaben abzugleichen. So entstehen maschinelle Fähigkeiten, die durch eine zielgerichtete funktionale Verknüpfung einzelner subsystemischer Komponenten bereitgestellt werden. Diese Komponenten werden im Rahmen dieser Arbeit als *Ressourcen* bezeichnet.

1.4 Aufgabenstellung

Aufgabenstellung dieser Arbeit ist es, ein holistisches, maschinen-verarbeitbares Modell zu erstellen und zu untersuchen, welches die für luftgestützte Aufklärungsaufgaben notwendigen systemischen Ressourcen an Bord eines Luftfahrzeuges erfasst, Abhängigkeiten der Ressourcen untereinander aufzeigt und diese mit entsprechenden inhärenten Fähigkeiten in Verbindung setzt.

1.4.1 Funktionale Vorgaben

Um die Anwendungsbreite des Modells zu demonstrieren, soll es in den folgenden Anwendungsfällen eingesetzt werden:

- **während der Missionsdurchführung**: Im Sinne des oben skizzierten Sensor- & Perzeptionsmanagements soll das Modell an Bord eines spezifisch ausgerüsteten Luftfahrzeuges zur Bestimmung einzelner geeigneter Aufklärungsmittel herangezogen werden. Spezifische Vorgaben (siehe Kapitel 3 für eine detaillierte Diskussion) in diesem Fall sind:
    - die Integrierbarkeit mit anderen Komponenten des S&PM,
    - die Aussagefähigkeit hinsichtlich möglicher Ressourcenkombinationen, sowie deren Limitationen und spezifischen funktionalen Anforderungen,
    - die Reaktionsfähigkeit bei Zustandsveränderungen,
    - die Einhaltung begrenzter Verarbeitungszeiten im Sinne typischer Systemzeiten.
- **vor der Missionsdurchführung**: Um eine allgemeinere, breitere Verwendbarkeit des Modells zu demonstrieren, soll es bereits vor der Durchführung der eigentlichen Mission anwendbar sein. Hierbei sollen zwei Ansätze verfolgt werden.
  <u>Aufgabengetrieben</u>: Unter Berücksichtigung gegebener Aufklärungsaufgaben soll das Modell in der Lage sein, eine geeignete UAV-Konfiguration zu ermitteln. Die Konfiguration beinhaltet u.a. die Sensornutzlast sowie die für die Aufgabe mitzuführenden Verarbeitungsfunktionen.
  <u>UAV-getrieben</u>: Hier steht die Aufklärungsaufgabe nicht fest, jedoch sind verschiedene UAVs und Ausrüstungen vorhanden, die miteinander kombiniert werden können. Das Modell soll dafür Aussagen treffen, welche möglichen Aufklärungsfähigkeiten insgesamt vorhanden sind.
  Besondere Herausforderungen hierbei sind:
    - effektive Handhabung der Modellgröße,
    - effiziente Strukturierung des Modells,
    - Optimierung der Kosten-Nutzen-Faktoren.

In beiden Ansätzen sollen mögliche Konfigurationen nach verschiedenen Faktoren, wie z.B. Kostenfaktor oder Effektivität, gewählt werden können.

1.4.2 Methodische Vorgaben

Zur Beschreibung eines Modells, das die eben erwähnten Aufgaben (Kapitel 1.4.1) erfüllt, kommen verschiedene Modellierungsmethoden in Frage, die in Kapitel 5 im Detail diskutiert werden. Von diesen erscheinen ontologiebasierte Ansätze als besonders interessant (siehe Kapitel 5). Die Möglichkeit, nicht nur Daten und Informationen, sondern auch Meta-Informationen semantisch in einer Ontologie zu beschreiben, erscheint an dieser Stelle erstrebenswert. Der Einsatz von Ontologien zur Beschreibung von voneinander abhängigen Informationen in anderen wissenschaftlichen Bereichen (z.B. dem *Semantic Web* [41]) hat sich als nützlich erwiesen. Zudem ermöglicht das Beschreiben eines Modells im Rahmen einer Ontologie, kontextbasierte Informationen nicht nur für Menschen (hier: Operateure), sondern auch für Maschinen (hier: dass sich an Bord des Luftfahrzeugs befindliche automatisierte Assistenzsystem) verständlich und interpretierbar darzustellen. Im Rahmen dieser Arbeit ergibt sich die Fragestellung, wie eine Ontologie gestaltet und die domänenabhängigen Informationen modelliert werden müssen, damit diese genutzt werden kann, um die funktionellen Vorgaben aus Kapitel 1.4.1 zu erfüllen.

1.4.3 Zusammenfassung

Aufgabe dieser Arbeit ist es, ein Ressourcenmodell zur Modellierung von Aufklärungsfähigkeiten auf Basis von Ontologiemethoden zu konzipieren, prototypisch zu implementieren und einer kritischen Betrachtung zu unterziehen. Merkmale, anhand derer eine erfolgreiche Umsetzung bewertet werden kann, sind dabei folgende:

- Berücksichtigung vorgegebener Kriterien zum Design einer Ontologie aus der Literatur (vgl. Kapitel 7.1: Strukturelle Bewertung).
- Beachtung und korrekte Umsetzung des Modells zur Erfüllung der in Kapitel 1.4.1 gestellten funktionalen Vorgaben (vgl. Kapitel 7.2: Operationelle Bewertung).
- Tolerable Verarbeitungszeiten des implementierten Softwaresystems an Bord des Luftfahrzeugs (vgl. Kapitel 7.3: Leistungsbewertung).

1.5 Wissenschaftlich-technischer Beitrag

Die vorliegende Arbeit leistet folgende innovative wissenschaftlich-technische Beiträge:

- Die Darlegung eines systemischen Ansatzes für ein holistisches Modell, welches zur Beschreibung von Ressourcen und damit verbundener Fähigkeiten von an Bord eines unbemannten Luftfahrzeugs installierten technischen Komponenten für sensorgestützte Aufklärungsaufgaben genutzt wird.
- Die Verwendung von Ontologien in einem Software-Framework zur Repräsentation von systemischen Zusammenhangswissen auf Subsystem- und Algorithmenniveau.

- Die Implementierung und Integration eines solchen Modells in ein Sensor- und Perzeptionsmanagementsystem zur automatisierten, aufgabengerechten und situationsadaptiven Auswahl von Aufklärungsmitteln.
- Die Untersuchung eines so integrierten Ressourcenmodells in einem dynamischen Kontext, welcher durch Variation der Umwelt- und Aufgabensituation das implementierte System veranlasst, diese Veränderungen während einer Flugmission festzustellen, die Ontologie dahingehend zu adaptieren und neue Aufgabenlösungen zu präsentieren.

Die aus der prototypischen Implementierung und Evaluierung gewonnenen Erkenntnisse sollen beim Design ähnlicher Ressourcenmodelle und dem Einsatz von Ontologien in der Luftfahrtdomäne unterstützen.

1.6 Gliederung der Arbeit

Kapitel Zwei „Grundlagen“ beschreibt zunächst den Ablauf einer Aufklärungsmission. Anschließend wird erläutert, welche Schritte bei einer Objekterkennung auf Basis von Bilddaten durchzuführen sind. In Kapitel 2.3 wird definiert, was im Rahmen dieser Arbeit unter Ressourcen und Fähigkeiten zu verstehen ist.

In *Kapitel Drei* werden die aus der Aufgabenstellung identifizierten Anforderungen an das Modell im Detail erläutert.

Das *vierte Kapitel* „Erweiterung des S&PM“ greift das Sensor und Perzeption Management System wie in Kapitel 1 beschrieben auf und erweitert das Konzept im Rahmen dieser Forschungsarbeit durch ein ontologiebasiertes Ressourcenmodell. Nachdem technische und funktionale Randbedingungen festgelegt wurden, wird auf den Aufbau, die Funktionsweise und die Komponenten des erweiterten Systems eingegangen.

Kapitel Fünf „Repräsentationsformen für das Ressourcenmodell“ beinhaltet Formalismen zur Repräsentation von Wissen. Nach einer Formalismenbewertung und -auswahl werden Beschreibungssprachen zur Umsetzung des gewählten Formalismus (Ontologien) vorgestellt.

Kapitel Sechs behandelt die Implementierungsdetails der im Rahmen der Konzepterweiterung entwickelten Softwarekomponenten und beschreibt die Kommunikation zwischen diesen.

Im *siebten Kapitel* „Evaluierung“ wird das im Rahmen dieser Arbeit entwickelte Softwaresystem zusammen mit der Ontologie in drei Experimentgruppen evaluiert. Zunächst findet eine strukturelle Bewertung der Ontologie statt. Anschließend wird das Softwaresystem in unterschiedlichen Szenarien in einer Simulationsumgebung operationell bewertet. Im dritten Verfahren werden Zeiten zu Abfragen der Ontologie durch zwei Verfahren untersucht.

Das *achte und letzte Kapitel* fasst die wesentlichen Erkenntnisse der wissenschaftlichen Arbeit zusammen, beschreibt die aus der Forschungsarbeit gewonnenen Ergebnisse und gibt einen Ausblick auf offene Forschungsfragestellungen sowie weiteres Vorgehen.

2

Grundlagen

Das Erstellen eines expliziten Ressourcenmodells hängt zum einen von den zu erfüllenden Anforderungen ab (vgl. Kapitel 1.4) und zum anderen von der Wahl geeigneter Methoden. Die Umsetzung eines solchen Modells ist mit unterschiedlichen Herausforderungen verbunden. Dieses Kapitel dient dem Zweck, ein grundlegendes Basiswissen und Verständnis für verschiedene Konzepte, Technologien und die damit verbundenen Anforderungen zur Verfügung zu stellen, die im Zusammenhang mit dem Thema dieser Arbeit stehen.

2.1 Ablaufbeschreibung einer Aufklärungsmission

Eine Aufklärungsmission besteht zumeist aus einer Sequenz von Aufgaben, die durchgeführt werden müssen. Diese können alternativ der Führung des UAVs oder dem Sensoreinsatz zugeordnet werden (siehe Kapitel 1.2).

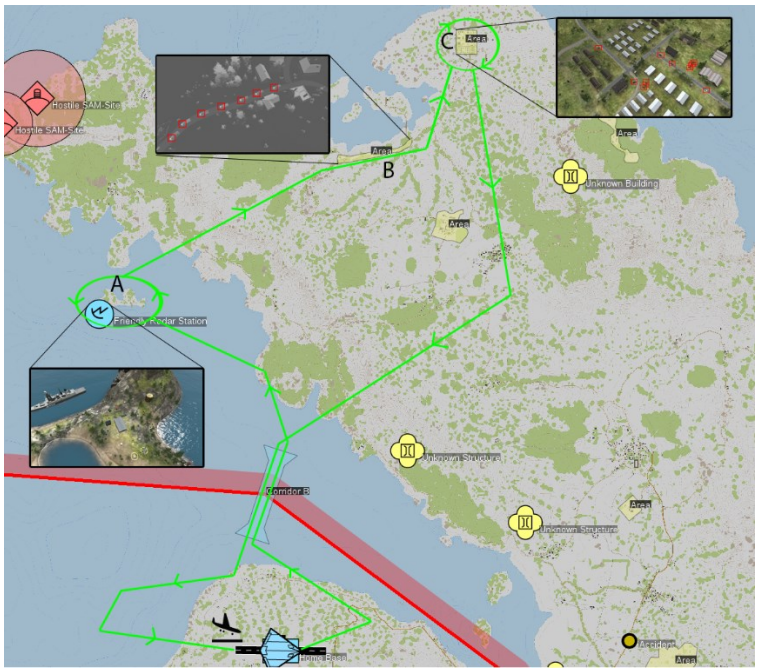


Abbildung 10: Beispielhafte Aufklärungsmission einschließlich Flugroute und durchzuführende Aufgaben

In Abbildung 10 ist eine Aufklärungsmission beispielhaft dargestellt. Zu sehen sind

- das Einsatzgebiet,
- der Flugplatz, auf dem die Mission gestartet bzw. beendet wird,
- ein Korridor, der passiert werden muss,
- die zu fliegenden Routen und
- drei Zielgebiete (A, B, C), in denen Aufklärungsaufgaben durchgeführt werden sollen.

Die folgende Beschreibung gibt einen Überblick über die wesentlichen Schritte und Aufgaben bei der Durchführung einer (militärischen) Aufklärungsmission. Dabei wird von einem UAS in konventioneller Missionsführung ausgegangen (vgl. Abbildung 7).

Nach Erhalt der Aufklärungsaufgabe gilt es zunächst die Einsatzmission vorzubereiten. Neben der Konfiguration und Ausrüstung des Fluggerätes selbst ist es notwendig, die sogenannte Missionsplanung durchzuführen. Diese beinhaltet im Wesentlichen die navigatorische Routenplanung. Neben den flugleistungsspezifischen Eigenschaften des Fluggerätes (z.B. optimale Geschwindigkeit und optimale Flughöhe) berücksichtigt die Routenplanung auch geografische Bedingungen wie Flugkorridore, Flugverbotszonen und Gefahrengebiete sowie zeitliche Einschränkungen. Nach dem Start führt das *Flight Management System* mit Hilfe des Autopiloten das UAV, gemäß der geplanten Streckenführung und unter Kontrolle des *Platform Operator,* in das Einsatzgebiet.

Dort führt die Besatzung die eigentliche Aufklärungsmission aus. Typische Aufgaben in diesem Kontext sind z.B. die Überwachung von Infrastrukturen (z.B. Gebäude oder technische Anlagen) oder etwa das Detektieren, Identifizieren und Verfolgen von statischen oder beweglichen Objekten (Personen, Fahrzeuge). Dazu führt wiederum der *Platform Operator* das Fluggerät auf einer für das Sensorsystem geeigneten Bahn, während der *Sensor Operateur* die eigentliche Aufklärungsaufgabe durchführt.

Dabei muss er zunächst entscheiden, welcher Sensor benutzt werden soll. Steht die Wahl fest, muss dieser eventuell parametrisiert werden. Soweit der Sensor über ein verstellbares Objektiv verfügt, muss die Zoomstufe an die Flughöhe, den aufzuklärenden Bereich oder die Objektgröße angepasst werden. Im Falle eines Infrarotsensors und in Bezug auf Tageszeit bzw. Umgebungstemperatur muss die Empfindlichkeit des Sensors angepasst werden. Atmosphärische Störungen bzw. Charakteristika können das Vorhaben zusätzlich erschweren.

Im nächsten Schritt müssen die Sensoren auf das Ziel ausgerichtet und die Sensoren gemäß der Flugbewegung nachjustiert werden. Diese wird heute bereits technisch unterstützt. So dämpft eine inertial gestützte Sensorhalterung („Gimbal“) die ungewollte Sensorauslenkung bei Böen. Weiterhin bietet die Einbeziehung von Positions- und Fluglagedaten die Möglichkeit, den Sensor über längere Zeit fix auf eine bestimmte Bodenposition auszurichten („Geo Lock“).

Ist das beabsichtigte Gebiet durch den Sensor erfasst worden, ist es Aufgabe des *Sensor Operators*, die o.g. Analysen und Beobachtungen anzustellen. Gegebenenfalls ist die Befliegung des Gebietes auch zu wiederholen und die Beobachtbarkeit (hypothetischer) der Zielobjekte zu

verbessern. Dies kann durch eine abermalige Umkonfiguration des Sensors, eine Veränderung der Flughöhe oder eine Anpassung der Beobachtungsrichtung (z.B. zur Kompensation des Sonnenstandes bzw. des Schattenwurfs) erfolgen.

Schließlich sind Beobachtungen im Bildmaterial zu annotieren und an die Auftraggeber zur detaillierteren Analyse weiterzuleiten.

2.2 Automatisierte Objekterkennung in Bilddaten

Wesentlicher Teil der Durchführung einer luftgestützten Aufklärungsmission ist der Erwerb und die Analyse des Sensordatenmaterials. Im Sinne der gesuchten Automatisierungslösungen kann dazu, wie bereits in Kapitel 1.3 angedeutet, auf das Methodeninventar der maschinellen *Bildverarbeitung* zurückgegriffen werden. Die dafür notwendigen Grundlagen sollen deshalb, soweit für den Inhalt dieser Arbeit notwendig, im Folgenden erläutert werden. Für detailliertere Ausführungen sind Standardwerke aus der Literatur zu empfehlen (z.B. [31], [47], [48]).

Werden Mustererkennungsalgorithmen auf Sensordaten angewendet, kommen grundsätzlich verschiedenste Quellen in Frage. Im Rahmen dieser Arbeit werden auszuwertende Daten betrachtet, die von einer elektrooptischen bzw. Infrarotkamera bereitgestellt werden. Die Bildverarbeitung oder Mustererkennung auf bildbasierten Daten hat das Ziel, Informationen aus (meist zweidimensionalen) Bilddaten zu extrahieren. Das Ergebnis sind wiederum Bilder bzw. Bildausschnitte oder eine Merkmalsmenge. Das Anwendungsspektrum ist dabei sehr breit. Medizin, Robotik, Automobilindustrie (z.B. Fahrassistenzsysteme), Überwachung (z.B. Gesichtserkennung) und eben luft- und raumfahrtgestützte Fernerkundung sind nur einige Beispiele.

2.2.1 Verarbeitungskette

Um Informationen aus Bilddaten zu extrahieren, werden eine Reihe mathematischer Operationen auf dem Bildmaterial durchgeführt. Eine ausgewählte Menge dieser Operationen in einer bestimmten Reihenfolge wird zu einem Bildverarbeitungsalgorithmus verkettet. Dabei spielt sowohl die Art als auch die Reihenfolge der Operationen eine Rolle. Je nachdem in welchem Format und in welcher Qualität (z.B. Auflösung, Belichtung, etc.) die Daten bereitgestellt werden und welche Informationen aus den Daten extrahiert werden sollen, müssen unterschiedliche Operationen durchgeführt werden (siehe Kapitel 2.2.2). Dabei haben die Operationen verschiedene Eigenschaften (z.B. Berechnungskomplexität), die je nach Verarbeitungsziel und Dateneigenschaften (z.B. Farbbild oder Bilder, die nur Graustufen enthalten), den Einsatz rechtfertigen oder ausschließen. Deshalb gibt es nicht den einen Bildverarbeitungsalgorithmus, der immer die gleichen Operationen in gleicher Reihenfolge ausführt und in jeder Situation auf allen Daten das gewünschte Ergebnis liefert, sondern eine Reihe von Operationen, die sich ähnlich sind, jedoch auf unterschiedlichen Daten beruhen und unterschiedliche Ergebnisse liefern. Während z.B. eine Operation sehr gut dazu geeignet ist, Merkmale aus Graustufenbilder zu extrahieren, ist eine andere Operation besser geeignet, um dieselben Merkmale aus einem Farbbild zu extrahieren. Die unterschiedlichen Operationen werden, je nachdem was sie tun und

wann sie eingesetzt werden, grob in folgende Klassen unterteilt (vgl. Abbildung 11). Das Ausführen der Operationen findet in der Regel, aber nicht ausschließlich, in der angegebenen Reihenfolge statt [49]:

- Datenakquisition,
- Vorverarbeitung (z.B. Farbraumkonvertierung),
- Segmentierung (z.B. Maskenbildung),
- Merkmalsextration (z.B. Kantendetektion),
- Klassifizierung (z.B. mit Methoden aus dem Bereich des maschinellen Lernens).

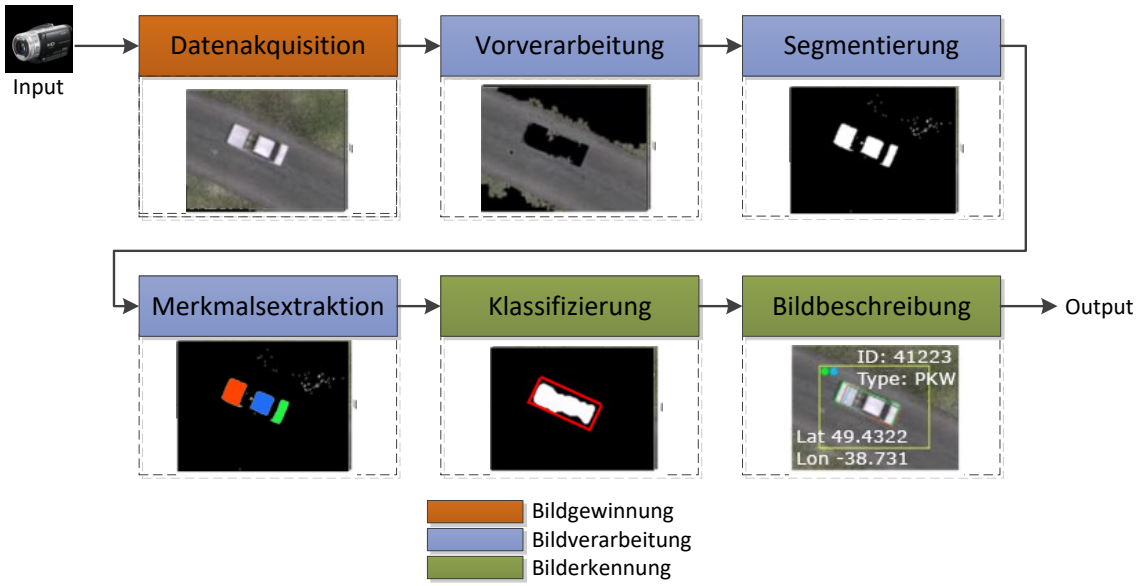


Abbildung 11 Verarbeitungskette der Bildverarbeitung [49]

Eine typische Anwendungssequenz der Operatoren lässt sich in drei Bereiche (Bildgewinnung, Bildverarbeitung und Bilderkennung) unterteilen und grob wie folgt erläutern: Zunächst wird eine Szene mittels eines Sensors erfasst bzw. gemessen. Während der *Datenakquisition* (Bilddiskretisierung) werden die Informationen auf eine für eine Maschine verwertbare zweidimensionale Pixelebene (Höhe x Breite) aufgespannt. Jeder Pixel kodiert dabei z.B. in einem Farbbild die Kanäle (Rot, Grün, Blau). Anschließend findet eine *Vorverarbeitung* auf den Daten statt, um die Informationen aufzubereiten bzw. das Bild zu verbessern (z.B. Belichtungssensitivität, Rotation, Rauschunterdrückung) oder eine Rohdatenreduktion durchzuführen. Im nächsten Schritt werden mittels *Segmentierung* interessierende Bereiche im Bild extrahiert, um dadurch den weiteren Berechnungsaufwand zu reduzieren. Anschließend findet die *Merkmalsextraktion* statt. Merkmale können z.B. markante Gradienten wie Kanten im Bild sein, die Teil der gesuchten Informationen sein könnten. Möchte man z.B. Fahrzeugdarstellungen aus einem (aus der Luft aufgenommenen) Bild extrahieren, könnte nach rechteckigen Strukturen im Bild gesucht werden. Im nächsten Schritt werden die Merkmale interpretiert und auf die gesuchten Informationen hin *klassifiziert*. Zuletzt werden die klassifizierten Daten beschrieben und mit weiteren Informationen (z.B. Geokoordinaten) versehen.

2.2.2 Beispiel einer variierenden Kette

Aus Kapitel 2.2.1 ist bekannt, dass eine Bildverarbeitung aus der Verkettung verschiedener Aufgaben besteht. Hier soll anhand eines Infrarotbildes (siehe Abbildung 12, in diesem Beispiel wird das farbkodierte Infrarotbild verwendet), auf dem sowohl ein Fahrzeug als auch Personen zu sehen sind, gezeigt werden, warum es sinnvoll sein kann, eine Verarbeitungskette nicht als Ganzes, sondern als eine Verkettung von Teilaufgaben zusehen.

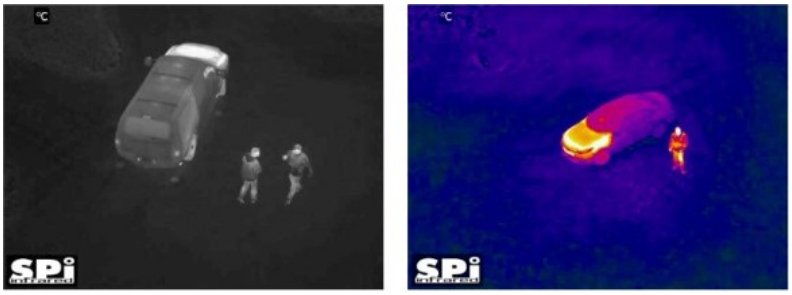


Abbildung 12: Beispiele für Personen und Fahrzeuge aus IR-Aufnahmen[11]. Links: Temperatur grauwertkodiert, Rechts: Temperatur farbkodiert.

Ziel ist es, aufgrund unterschiedlicher Geometrie sowohl die Personen als auch das Fahrzeug zu detektieren und als solche zu identifizieren. Hierfür sind zwei unterschiedliche Bildverarbeitungsalgorithmen notwendig, die jedoch gemeinsame Verarbeitungsschritte aufweisen.

Da die Detektion beider Objekte auf derselben Datenquelle basiert, sind die Verarbeitungsschritte bis auf die *Merkmalsextraktion* und *Klassifizierung* identisch. Abbildung 13 veranschaulicht beide Algorithmen (blaue Pfeile für die Detektion des Fahrzeugs, gelbe Pfeile für die Detektion der Personen) und hebt die identischen Operationen hervor.

In beiden Fällen werden die Daten ins RGB-Format konvertiert. Anschließend werden eine Rauschunterdrückung und eine Glättung durch einen Medianfilter durchgeführt. Bei der Segmentierung werden Masken auf Basis von Pixelwerten (hohe Pixelintensität) gebildet. Der erste Unterschied tritt bei der Merkmalsextraktion auf. Je nachdem was detektiert werden soll, werden unterschiedliche Merkmale extrahiert. Im nächsten Schritt kann beispielhaft angenommen werden, dass Methoden des maschinellen Lernens eingesetzt werden, um in einem Fall Klassifikatoren zu trainieren, die in der Lage sind, in einem Fall Fahrzeuge aufgrund bestimmter Merkmale zu klassifizieren und im anderen Fall Personen. Der letzte Schritt ist für beide Aufgaben wiederum identisch. Die erkannten Personen bzw. Fahrzeuge werden mit Metadaten, wie z.B. einer ID der Erkennung oder Positionskoordinaten, versehen.

[11] Bildquelle: https://tinyurl.com/ycn7azuv, 18.04.2018

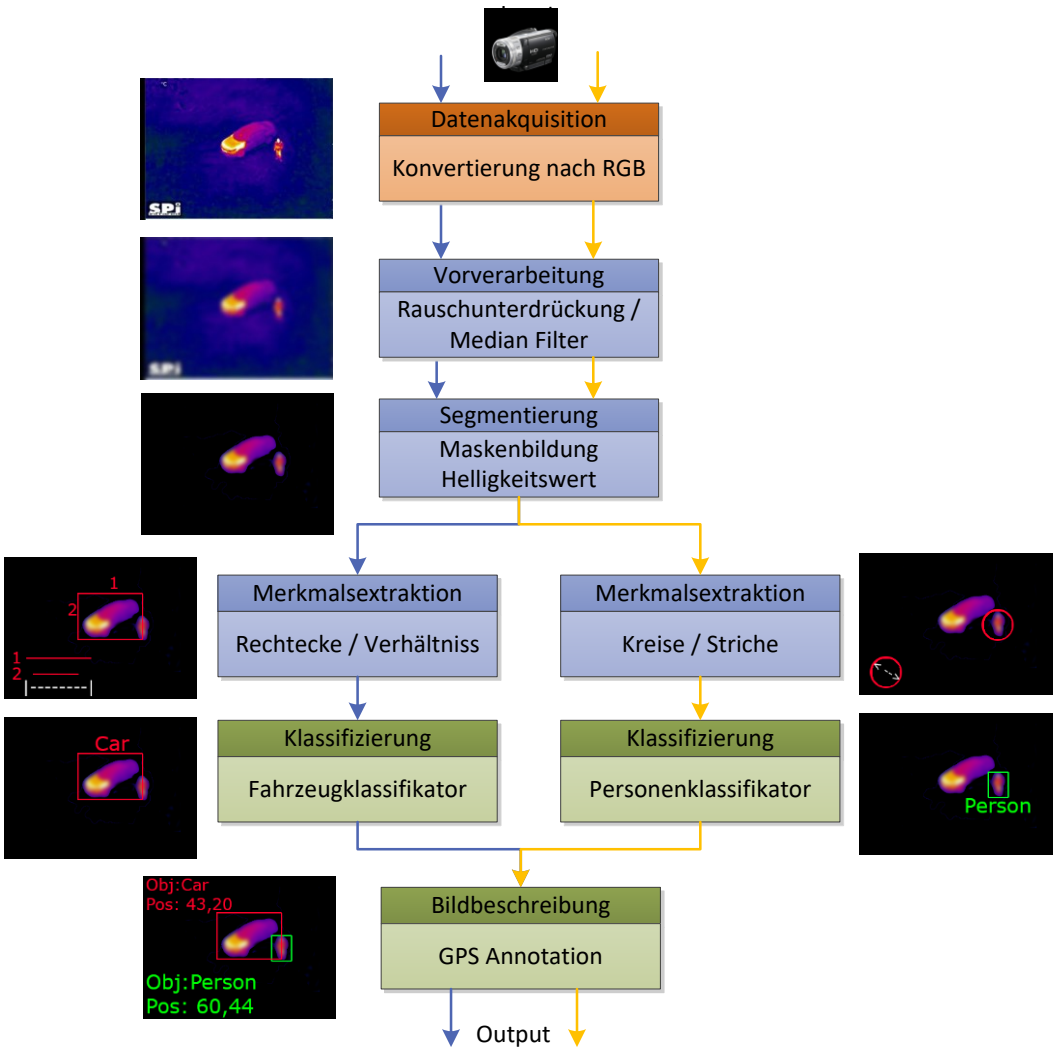


Abbildung 13: Zwei Bildverarbeitungsalgorithmen zur Detektion von Personen und Fahrzeugen

2.3 UAV-bezogene Ressourcen und Fähigkeiten

Die Begrifflichkeiten *Ressource* und *Fähigkeit* sind im Rahmen dieser Arbeit von wesentlicher Bedeutung. Sie bezeichnen Konzepte, mit denen ein automatisiertes System in die Lage versetzt werden soll, zu identifizieren, ob, wann und wie eine Aufklärungsaufgabe durchgeführt werden kann. Im Folgenden sollen diese beiden Begriffe näher im Kontext erläutert und die damit verbundenen Probleme und Fragestellungen erörtert werden.

2.3.1 Ressourcen

Ressource ist ein breitgefächerter Begriff, dessen Bedeutung je nach Fachgebiet und Kontext auf verschiedene Arten interpretiert und definiert werden kann [50]. Zu den Gebieten, die sich mit Ressourcen bzw. deren Modellierung auseinandersetzen, gehören unter anderem das Personalwesen [51], Logistik, Betriebswirtschaftslehre und Produktionsplanung. Nicht für jedes Fachgebiet gibt es für diesen Begriff eine fest vorgeschriebene Definition.

In der Allgemeinsprache ist der Begriff Ressource (aus dem Französischen für „Mittel“ bzw. „Hilfsmittel“) definiert als

> *„natürlich vorhandener Bestand von etwas, was für einen **bestimmten Zweck**, besonders zur Ernährung der Menschen und zur wirtschaftlichen Produktion, [ständig] benötigt wird“.* [52]

In den verschiedenen wissenschaftlichen Disziplinen schließt der Begriff sowohl materielle wie auch immaterielle Dinge ein. Zu den materiellen Dingen zählen z.B. Rohstoffe [53], Güter und Geldmittel [54]. Immaterielle Dinge umfassen z.B. Zeit, Informationen und Wissen [55]. Eine weitere Ressource, die an dieser Stelle besonders herausgehoben werden muss, ist der Mensch selbst. Betrachtet man den Menschen aus der Sicht des Personalwesens, so sind die Fähigkeiten, Charaktereigenschaften, Arbeitsleistung und Bildung des Menschen Eigenschaften, die ihn als Ressource beschreiben und bewertbar machen.

In der Informationstechnologie bzw. genauer in der Informatik wird der Begriff IT-Ressource verwendet. Gemeint sind virtuelle (z.B.: Prozesse, Rechenzeit, Speicherverfügbarkeit) oder physikalische (z.B.: Arbeitsspeicher, Kabel, Monitor) Komponenten, die als Betriebsmittel Teil einer IT Infrastruktur sind. Zusätzlich zählen auch Softwaretools und -algorithmen, Lizenzen und Datenbestände zu IT- Ressourcen.

Im Rahmen dieser Arbeit werden Elemente aus verschiedenen Kategorien und Bereichen unter dem Begriff der Ressource zusammengefasst. Hierzu gehören sowohl materielle als auch immaterielle Komponenten. Das gemeinsame Merkmal ist ihr funktioneller Beitrag bei der Erfüllung einer Aufklärungsleistung. Abbildung 14 visualisiert diese Komponenten in einem vereinfachten Schema und deutet auch bereits erste funktionale Bezüge an [46].

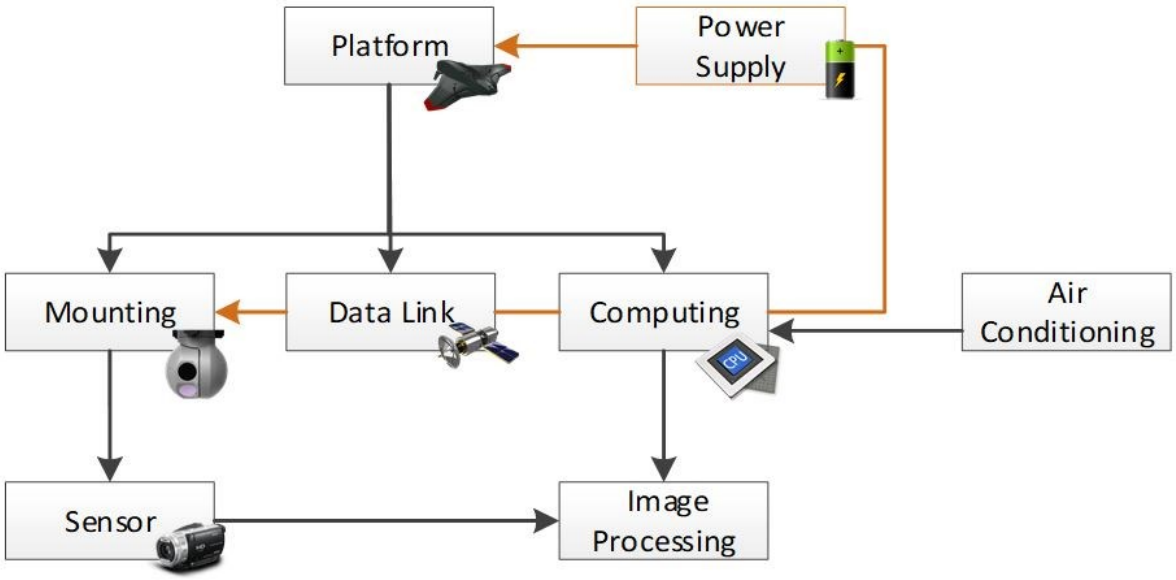


Abbildung 14: Ressourcen im Sinne des Sensor- und Perzeptionsmanagement

2.3.1.1 Ressourcenelemente

Im Sinne der Informationsverarbeitung erfassen **Sensoren** die von der Umwelt ausgesendete elektromagnetische Strahlung und stellen die Informationen in digitalisierter oder analoger Form zur Weiterverarbeitung zur Verfügung. Bildverarbeitungsalgorithmen (**Image Processing**) werden auf Rechnerkomponenten (**Computer**) ausgeführt, bereiten die Sensorinformationen auf und werten sie mit SW-Algorithmen aus. Die so erzielten Ergebnisse werden gegebenenfalls mittels einer Datenverbindung (**Data Link)** an den Boden übermittelt. Aus Sicht der mechanischen Interaktion sind die Sensoren über Sensoraufnahmen (**Mounting**) entweder fix oder beweglich verbaut, welche wiederum in das Luftfahrzeug (**Platform**) integriert sind. Das Luftfahrzeug hat daneben die Aufgabe, durch seine prinzipiellen Eigenschaften die Sensorik entsprechend der Aufgabe zu positionieren. Des Weiteren versorgen die Ressourcenelemente **Power Supply** und **Air-Conditioning**, soweit erforderlich, andere Komponenten mit elektrischer Energie respektive Kühlleistung. Abbildung 15 zeigt einige Beispiele für Ressourcen im Sinne des S&PM.

Abbildung 15: Beispiele für Ressourcen (Platform, Computer, Gimbal, Sensor)

Zur besseren Strukturierung und Beschreibung der Ressourcen können diese unterschiedlichen Systembereichen zugeordnet werden. In der Luftfahrtdomäne werden z.B. **ATA**-Kapitel (***Air Transport Association***) [56] benutzt, um die einzelnen Systeme eines Flugzeugs in Gruppen einzuteilen. Zur Gewährleistung der Interoperabilität[12] werden die hier beschriebenen Ressourcen ebenfalls den verschiedenen Gruppen der ATA-Kapitel zugeordnet (siehe Kapitel 6.2.1). Diese Zuordnung stellt sicher, dass diese, unabhängig vom Anwender oder Verwendungszweck der Ressourcen, in der Luftfahrtdomäne immer im selben Kontext beschrieben werden.

2.3.1.2 Ressourcenbeziehungen

Um eine Aufgabe im Sinne einer Luftaufklärungsmission automatisiert durchzuführen, gilt es herauszufinden, welche Ressourcen für eine bestimmte Aufgabe konkret notwendig sind und welche Abhängigkeiten sie untereinander haben. Ein Bildverarbeitungsalgorithmus z.B. kann nur dann richtige Ergebnisse liefern, wenn er die Daten von einem geeigneten Sensor bekommt. Es stellt sich die Frage, welche funktionellen Beiträge Ressourcen generell bzw. spezifisch sind, welche Anforderungen für einen gewinnbringenden Einsatz an sie gestellt werden, wie sie miteinander zu kombinieren sind und wo eventuelle Beschränkungen liegen. Allgemein müssen Schnittstellen gefunden und definiert werden. Schließlich muss definiert werden, wie solche

[12] Fähigkeit unterschiedlicher Systeme, möglichst nahtlos zusammenzuarbeiten.

Ressourceneigenschaften innerhalb eines einheitlichen Modells beschrieben und wie Ressourcen aus verschiedenen systemischen bzw. technischen Domänen miteinander so in Einklang gebracht werden, dass durch die Verknüpfung ein Mehrwert im Sinne der Missionsaufgabe entsteht.

Um die genannten Punkte technisch erfüllen zu können, bedarf es einer einheitlichen Beschreibung und einer Sprache bzw. Syntax, damit Informationen aus verschiedenen technischen Domänen auf die gleiche Art und Weise interpretiert werden können. Hierfür gibt es in der Literatur verschiedene Konzepte und Ansätze. Einige von ihnen werden im späteren Verlauf dieser Arbeit (siehe Kapitel 5.1) näher betrachtet. Um identifizieren zu können, welche Ressourcen für eine Aufgabe notwendig sind, werden die von einer Ressource zur Verfügung gestellten Fähigkeiten bzw. für eine Aufgabe notwendigen Fähigkeiten beschrieben. Solche Fähigkeitenmodelle können anschließend genutzt werden, um festzustellen, welche Ressourcen für eine Aufgabe notwendig sind bzw. wie die Ressourcen miteinander verknüpft werden müssen. Aus der Fähigkeitenbeschreibung ergeben sich Abhängigkeiten, da weitere Ressourcen z.B. nur dann genutzt werden können, wenn weitere Fähigkeiten, die wiederum von anderen Ressourcen bereitgestellt werden, zur Verfügung stehen. Damit z.B. ein Algorithmus, der auf Basis von thermischen Signaturen Objekte (z.B. Personen am Boden) erkennt, genutzt werden kann, muss zuvor sichergestellt werden, dass ein Sensor zur Verfügung steht, der in der Lage ist, diese Informationen bereitzustellen. Andernfalls ist dieser Algorithmus, sprich diese Ressource, mit der Fähigkeit, ein bestimmtes Objekt zu detektieren, zwar vorhanden, kann jedoch aufgrund der fehlenden Vorbedingung nicht benutzt werden. Weiterhin reicht die bloße Existenz eines Sensors, der thermische Signaturen detektieren kann, nicht aus. Es muss zudem gewährleistet werden, dass der Sensor bewegt werden kann, um Suchbewegungen auszuführen.

Abbildung 16 veranschaulicht die Abhängigkeit der genannten Ressourcen und die damit verbundenen Fähigkeiten. Die Beziehungen zwischen den Ressourcen sind bereits aus Abbildung 14 bekannt. Zusätzlich wird verdeutlicht, wie mittels Fähigkeiten sichergestellt wird, dass ein Bildverarbeitungsalgorithmus in der Lage ist, Personen zu erkennen. Für die Erkennung werden thermische Signaturen benötigt, die vom Sensor bereitgestellt werden. Damit die Erkennung stattfinden kann, ist eine bestimmte Perspektive notwendig. Diese wird durch die Fähigkeit, die Sensororientierung zu verändern, sichergestellt. Die Fähigkeit wird durch das Mounting bereitgestellt.

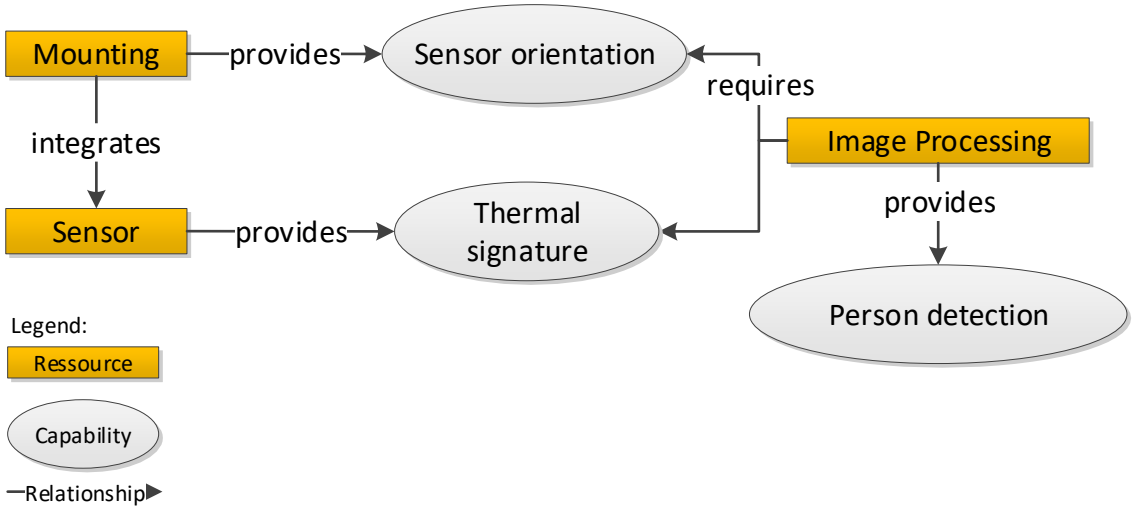


Abbildung 16: Zusammenhang zwischen Ressourcen und Fähigkeiten. Ressourcen werden durch Rechtecke, Fähigkeiten durch Ellipsen repräsentiert.

2.3.2 Fähigkeiten

Im vorhergehenden Kapitel wurde der Begriff *Fähigkeit* (*capability*) bereits mit Ressourcen in Zusammenhang gebracht (siehe [57] für allgemeine Definitionen des Begriffs). In der Allgemeinsprache ist eine Fähigkeit definiert als eine

> *„durch bestimmte Anlagen[und] Eigenschaften geschaffene Möglichkeit,* ***gewisse Funktionen zu erfüllen****, gewissen Anforderungen zu genügen****, etwas zu leisten****“* [59].

Eine Definition, die im Zusammenhang mit Ressourcen und deren Fähigkeiten aus dem Bereich der Geschäftsprozesse genannt wird und für diese Arbeit präziser zutrifft, gibt [60]:

> *„Eine Fähigkeit wird durch die Möglichkeit zur zielgerichteten Anwendung von Kenntnissen definiert,* ***die der Lösung einer Aufgabe (eines Problems) dienen****.*
> *Fähigkeiten können sowohl geistiger als auch motorischer Natur sein und* ***andere Fähigkeiten erfordern****. Ihre Qualität lässt sich durch Verknüpfung mit einem Fähigkeitsniveau bestimmen.“* [60]

Im vorliegenden Kontext soll der Begriff spezifischer erläutert werden. Der Begriff *Fähigkeit* [58] bezeichnet die quantitative oder qualitative Leistung einer Ressource, die durch das automatisierte System zur Verfügung gestellt wird. Die Beschreibung von Fähigkeiten verfolgt dabei den Zweck,

- Abhängigkeiten zwischen Ressourcen zu dokumentieren (siehe Kapitel 2.3.1.2) und
- aufzuzeigen, welche Aufklärungsaufgaben letztlich durchgeführt werden können.

Sind z.B. die Fähigkeiten vorhanden, *thermische Signaturen wahrzunehmen* und *Personen zu detektieren*, so kann daraus abgeleitet werden, dass die Aufgabe „*Personen bei Nacht detektieren*“ durchgeführt werden kann.

Kann durch Verknüpfung verschiedener Fähigkeiten die Lösung einer Aufgabenstellung beschrieben werden (vgl. Abbildung 16), können unmittelbar auch die jeweils notwendigen Ressourcen festgelegt werden. Soll diese Verknüpfung ausgeführt werden, muss geprüft werden, welche spezifischen Anforderungen und Beschränkungen die Ressourcen aufweisen.

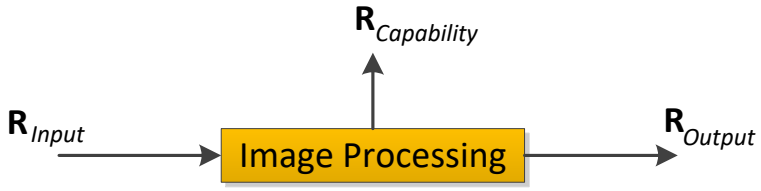


Abbildung 17: Beschreibung einer Ressource

Die Fähigkeitenbeschreibung einer Ressource ist in zwei Teile unterschieden. Um die Abhängigkeit zwischen Ressourcen zu beschreiben, ist neben der Fähigkeit selbst noch eine Beschreibung des Formates der Eingangs- bzw. Ausgangsdaten notwendig (vgl. Abbildung 17, R_{Input} und R_{Output}). Um festzustellen, welche Ressourcen notwendig sind, um eine Aufklärungsaufgabe durchzuführen, reicht die Fähigkeitenbeschreibung $R_{Capability}$. Eine Beschreibung einer Ressource R^1 (Bildverarbeitungsalgorithmus zum Erkennen von Fahrzeugen) ist im Folgenden gegeben:

$$R^1{}_{Capability} = \{Methode: Detektieren, Objekt: Fahrzeug\}$$

$$R^1{}_{Input} = \{Format: RGB, Typ: ROI\}$$

$$R^1{}_{Output} = \{Format: Koordinate, Typ: POS(Lat, Lon)\}$$

Um festzustellen, ob diese Ressource R^1 für eine bestimmte Aufklärungsaufgabe prinzipiell genutzt werden kann, muss ebenfalls die Beschreibung dieser Aufgabe, wie sie im Folgenden gezeigt wird, betrachtet werden.

$$Aufgabe_{PKW} = \left\{ \begin{matrix} \boldsymbol{Methode: Detektieren,} \\ \boldsymbol{Objekt: Fahrzeug,} \\ Typ: PKW, \\ Ort: Straße, \\ Uhrzeit: 23:00 \end{matrix} \right\}$$

Aus der Beschreibung der Aufgabe und der Ressource kann geschlossen werden, dass diese Ressource für die Aufgabe in Frage kommt, weil die *Methode* und das *Objekt* in den Anforderungen der Aufgabenbeschreibung $Aufgabe_{PKW}$ identisch mit der Fähigkeit $R^1{}_{Capability}$aus der Beschreibung der Ressource R^1 ist (auf die Anforderungen Typ, Ort und Uhrzeit soll an

dieser Stelle nicht weiter eingegangen werden.). Bevor die Ressource jedoch verwendet werden kann, muss sichergestellt werden, dass es andere Ressourcen gibt, die die Vorbedingung von R^1, also ${R^1}_{Input}$ erfüllen. Das heißt, es muss eine Ressource R^x geben, die folgende Nachbedingung ${R^x}_{Output}$ besitzt:

$$ {R^x}_{Output} = \{Format: RGB, Typ: ROI\} $$

Um eine Aufgabe mittels einer Ressource zu erfüllen, reicht es nicht aus, dass es eine Ressource gibt, deren Fähigkeit sich mit der angeforderten Fähigkeit der Aufgabenbeschreibung deckt. Die Vor- und Nachbedingungen der Ressource sowie der damit verbundenen Ressourcen müssen ebenfalls erfüllt sein. Die Vor- bzw. Nachbedingungen einer Ressource müssen dabei nicht zwingend von einer einzigen Ressource, sondern können ebenfalls von mehreren Ressourcen erfüllt werden. So kann aufgrund der notwendigen Fähigkeiten aus einer einfachen Aufgabenbeschreibung eine komplexe Verkettung entstehen (siehe Abbildung 18). Dabei muss es nicht zwingend eine gültige Kette geben, die genutzt wird, um die Aufgabenbeschreibung zu erfüllen, sondern es können hierfür auch mehrere redundante Ketten existieren.

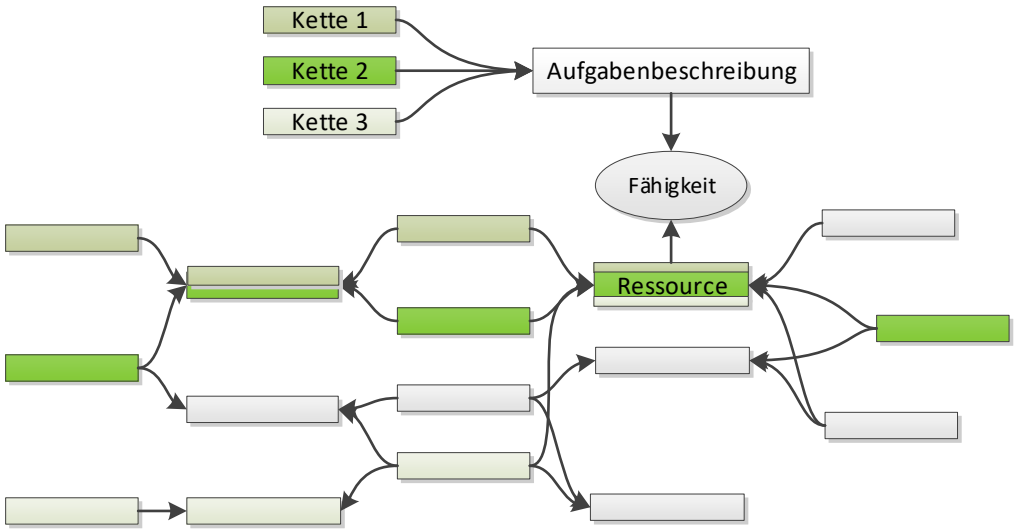


Abbildung 18: Verkettung von Ressourcen aufgrund von Vor- und Nachbedingungen

Um das geschilderte Paradigma zu implementieren, muss eine geeignete Repräsentationsform und ein Beschreibungsformat gefunden werden, sodass Fähigkeiten in einer ganzheitlichen Art und Weise abgebildet werden können. Die Ressourcenfähigkeit sowie der notwendige In- bzw. Output werden in Kapitel 4.4 im Detail besprochen.

3

Anforderungen an das Modell

Nachdem die für das Thema dieser Arbeit notwendigen Grundlagen erörtert wurden, kann im Folgenden im Detail auf die in Kapitel 1.4 (Aufgabenstellung) aufgestellten funktionalen und methodischen Vorgaben an das Modell eingegangen werden.

3.1 Integrierbarkeit in das bestehende Systemkonzept

In Kapitel 1.3 wurde die Aufgabe und Wirkweise des *Sensor- und Perzeptionsmanagementsystem* an Bord eines unbemannten Luftfahrzeugs erläutert. Weiterhin wurde im Rahmen früherer Forschungstätigkeiten bereits ein solches System diskutiert. Eine erste Version wurde implementiert und im Rahmen der vorhandenen Funktionen erfolgreich erprobt. Das im Rahmen dieser Arbeit zu entwickelnde Ressourcenmodell soll in das bestehende System integriert und um die vom Ressourcenmodell bereitgestellten Funktionalitäten erweitern werden. Dadurch sollen einige konzeptuelle Lücken geschlossen werden.

Dabei ist darauf zu achten, dass bereits bestehende Funktionalitäten beibehalten und neue Funktionalitäten innerhalb der bestehenden Architektur bereitgestellt werden. Um dies zu gewährleisten, soll die bereits bestehende Systemarchitektur angepasst werden. Durch die Integration des Ressourcenmodells wird der Funktionsumfang des Gesamtsystems zwar erweitert, die zu erwartende steigende Komplexität soll jedoch den Wartungsaufwand des Systems nicht negativ beeinflussen. Im folgenden Kapitel 4 wird im Rahmen der Systemerweiterung spezifischer auf die Integrierbarkeit eingegangen.

3.2 Aussagefähigkeit hinsichtlich möglicher Ressourcenkombinationen

Wie bereits in Kapitel 2.1 gezeigt, besteht eine Aufklärungsmission aus verschiedenen Aufgaben, die von einem sich an Bord befindlichen automatisierten Assistenzsystem (vgl. Kapitel 3.1) nacheinander abgearbeitet werden müssen. In Abhängigkeit von unterschiedlichen Kriterien wie der verfügbaren Sensorik und Hardware, der Beschaffenheit des Zielgebietes oder z.B. der Wetterlage und Tageszeit, muss mittels des Modells für eine spezifische Aufgabe ein passender Lösungsweg gefunden werden. Die Bereitstellung solcher zum Teil auch redundanter Lösungsstrategien wird vor allem durch unterschiedliche Verarbeitungsalgorithmen (z.B. für das Detektieren von Fahrzeugen mit Hilfe von *Templates* oder auf Basis von *Feature-Extraction*) sowie das Mitführen mehrerer Sensoren mit unterschiedlichen Einsatzcharakteristiken gewährleistet.

Diese Aussagefähigkeit spielt sowohl während der Missionsdurchführung (z.B. im Fall von Systemausfällen, siehe Kapitel 3.3) als auch in der Missionsvorbereitung während der Planungsphase eine entscheidende Rolle. In der Planungsphase kann die Aussagefähigkeit, wie bereits erwähnt, aufgabengetrieben (Top-Down: welche UAV-Konfiguration ist ausgehend von einer spezifischen Aufgabe zu wählen) oder UAV-getrieben (Bottom-Up: welche Aufgaben können mit einer spezifischen UAV-Konfiguration durchgeführt werden) genutzt werden. Weiterhin kann die Aussagefähigkeit vor der Missionsdurchführung dazu verwendet werden, um eine Aussage über Kosten-Nutzen-Faktoren zu bekommen. Im Falle zweier konkurrierender UAV-Konfigurationen kann so z.B. die günstigere Variante gewählt werden.

Aufgrund der limitierten Leistungskapazitäten an Bord eines unbemannten Luftfahrzeugs ist es sinnvoll, wichtige Ressourcen (z.B. Sensordatenströme und prozessierte Informationen) mehrfach an unterschiedlichen Stellen wiederzuverwenden. Deshalb werden komplexe Aufgaben in Teilaufgaben separiert, sodass gleiche Verarbeitungsschritte nicht mehrfach ausgeführt werden müssen.

Aus Kapitel 2.3 ist bekannt, dass verschiedene Ressourcen notwendig sind, um eine aufgaben- und situationsgerechte Lösungsstrategie zu generieren. Veranschaulichen lässt sich das als gerichteter Graph mit Knoten und Kanten (Abbildung 19). Dabei repräsentieren Knoten einzelne Ressourcen und Kanten kompatible Ressourcenkombinationen, deren Richtung funktionale Beiträge bzw. Informationsflüsse darstellen.

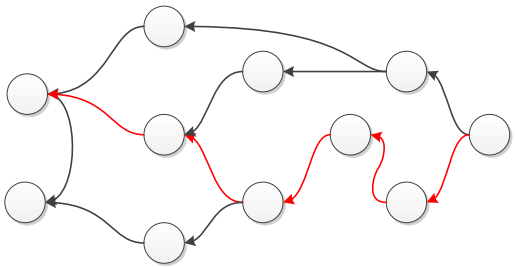


Abbildung 19: Schematischer Perzeptionsgraph von notwendigen Ressourcen für eine Gesamtfähigkeit. Der rote Pfad markiert eine Kette (aus mehreren möglichen), die gewählt werden kann, um die Aufgabe durchzuführen.

Der Graph in Abbildung 19 zeigt Lösungsstrategien für ein und dieselbe Aufgabe. Das heißt, die einzelnen Ketten im Graph repräsentieren verschiedene Lösungsstrategien für ein Problem bzw. eine Aufgabe, wobei sie sich allerdings in der Qualität der Ergebnisse, der Anzahl der gebundenen Ressourcen sowie in der Verarbeitungsdauer unterscheiden. Der rote Pfad kennzeichnet dabei eine mögliche Traversierung im Sinne einer solchen redundanten Kette.

Um dieses Ergebnis zu erhalten, muss das Ressourcenmodell gewährleisten, dass alle relevanten Informationen, die zur Identifizierung einer Lösung dienen, darin abgelegt werden können. Dies betrifft nicht nur Informationen, mit denen die Reihenfolge und Abhängigkeit der Ressourcen identifiziert werden kann, sondern auch Informationen rund um Limitationen und spezifische funktionale Anforderungen an die Ressourcen. Neben den in Kapitel 2.3.2 erwähnten Informationen, die zur Verknüpfung von Ressourcen genutzt werden, betrifft dies auch numerische Daten zur Parametrisierung von Ressourcen, wie z.B. Sensoren oder Algorithmen.

3.3 Reaktionsfähigkeit bei Zustandsveränderungen

Im Fall von Systemausfällen oder sich ändernden Umweltbedingungen soll das Ressourcenmodell herangezogen werden, um die Auswirkungen dieser Veränderungen auf das Gesamtsystem und die Aufgabenlösung bestimmen zu können. Hierbei soll die vorgestellte Aussagefähigkeit (Kapitel 3.2) des Modells und die damit verbundene Auslegung redundanter Lösungsstrategien dazu dienen, alternative Lösungen in Abhängigkeit der Veränderung (Systemausfall einer Komponente, veränderte Umweltbedingung oder Änderung der Aufgabe) zu identifizieren.

Die Veränderung im System (Strategiewechsel) muss dem an Bord des UAV-befindlichen Assistenzsystem mitgeteilt werden können, um z.B. im Falle eines Datenlinkverlustes die Aufgabe dennoch ausführen zu können.

3.4 Einhaltung begrenzter Verarbeitungszeiten

Das zu erstellende Modell soll typische Luftaufklärungsmissionen beispielhaft abdecken. Im Sinne eines Proof-of-Concept sollen ausgewählte Ressourcenarten und UAV-Typen betrachtet werden. Die Auswahl orientiert sich dabei an der Notwendigkeit, die Eigenschaften des gewählten Lösungsansatzes aufzuzeigen und zu bewerten. In diesem Sinne sollen Vereinfachungen und Verallgemeinerungen in einzelnen Bereichen als zulässig erachtet werden.

Gerade, aber nicht nur im Falle von Zustandsveränderungen, wie z.B. Systemausfällen, spielt es eine entscheidende Rolle, wie lange das System benötigt, um (1) die Zustandsveränderung wahrzunehmen und (2) entsprechende Maßnahmen zu identifizieren und umzusetzen. Daraus entsteht eine Leistungsanforderung für eine gesamtsystemgerechte Verarbeitungs- und Antwortzeit bei der Erstellung und Nutzung des Ressourcenmodells. Zwar gibt es keine harten Echtzeitanforderungen, jedoch muss ein gewisser zeitlicher Rahmen eingehalten werden, damit eine Mission erfolgreich durchgeführt werden kann. Dies gilt nicht nur für eine außerplanmäßige Veränderung während der Mission, sondern auch für geplante Aufgaben, wie z.B. das Identifizieren von Objekten in einem vorgegebenen Gebiet. Das Identifizieren der Fähigkeiten

und die damit verbundenen Restriktionen und Anforderungen sowie das Bereitstellen von Lösungsstrategien zum Durchführen von Perzeptionsaufgaben dürfen den Missionsablauf des Luftfahrzeugs nicht beinträchtigen.

3.5 Domänenübergreifende Konzeptrepräsentation

In Kapitel 2.3.1 wurde mit der Einführung der ATA-Kapitel bereits eine Möglichkeit vorgestellt, wie das Ressourcenmodell durch die Zuordnung der Ressourcen zu verschiedenen Luftfahrtsystemdomänen strukturiert werden kann. Aus Kapitel 1.4 ist bekannt, dass das Modell in zwei verschiedenen Anwendungsbereichen (vor der Mission und während der Mission) genutzt werden soll. Aus diesen zwei Anwendungen ist zu schließen, dass es von mindestens zwei unterschiedlichen Nutzern verwendet wird (UAV-Operateur und Missionsplaner). Betrachtet man das Modell genauer, stellt man fest, dass sich Teile davon mit weiteren Domänen überschneiden. Zum Beispiel gibt es für die Sensorik und Bildverarbeitungsalgorithmen ganz eigene Anwendungsgebiete (siehe Kapitel 5.3.3). Aus den verschiedenen Nutzergruppen, Domänen und Anwendungsgebieten ergibt sich das Problem, dass sich der Fokus von Nutzer zu Nutzer bzw. Anwendung unterscheidet (Sensor zum Messen der Raumtemperatur, Sensor zum Generieren von Bildinformationen). Weiterhin werden dieselben Konzepte je nach Domäne mit verschiedenen Begrifflichkeiten (Sensor vs. Sensorik) beschrieben. Zuletzt muss eine gewisse Flexibilität bereitgestellt werden, damit bestehende Konzepte erweitert werden können. Abbildung 20 veranschaulicht die Betrachtung eines bestimmten Konzeptes aus Sicht unterschiedlicher Nutzer.

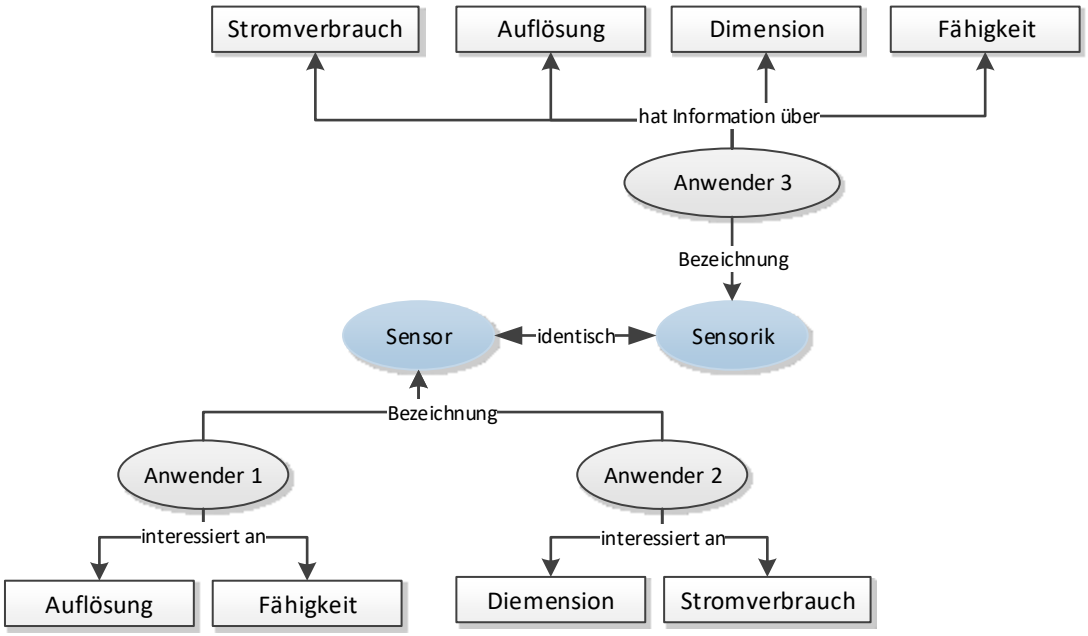


Abbildung 20: Beschreibung des Konzeptes "Sensor" aus Sicht verschiedener Anwender

Damit das hier vorgestellte Ressourcenmodell domänenübergreifend einer möglichst breiten Anwenderschicht bereitgestellt werden kann, muss zum einen eine gemeinsame Syntax bzw.

Beschreibungssprache mit einem vordefinierten Vokabular gefunden werden und zum anderen sind Verknüpfungen zu bereits bekannten Konzepten erforderlich.

Um dies zu erreichen, sollen die verschiedenen Subsysteme des Ressourcenmodells in eigenständigen domänenspezifischen *Wissenspaketen* bzw. Modellen beschrieben werden. Je nachdem, welche Subsysteme ein unbemanntes Luftfahrzeug letztlich mit sich führt, sollen diese Modelle der einzelnen Systeme zusammengeführt und an einer zentralen Stelle verknüpft werden. Ein entsprechender Fusionsmechanismus soll identische Beschreibungen bzw. Terme als solche identifizieren und damit die Subsystemmodelle untereinander verknüpfen. Dabei gilt es zu beachten, dass die Komplexität eines solchen holistischen Modells, das alle vorhandenen Informationen miteinander verknüpft und die Daten in Relation setzt, mit jeder neuen Ressource ansteigt.

4

Erweiterung des S&PM

In der Problemstellung dieser Arbeit (Kapitel 1.3) wurde die Notwendigkeit eines *Sensor- & Perzeptionsmanagements* diskutiert. Um die in Kapitel 1.4 definierten Vorgaben erfüllen zu können, soll das S&PM um das im vorhergehenden Kapitel beschriebene Ressourcenmodell erweitert werden. Das in [39] vorgestellte S&PM ist bereits in der Lage, mit Sensoren und Bildverarbeitungsketten umzugehen. Nun soll das Paradigma des S&PM um die Betrachtung weiterer Ressourcen, die für eine Luftaufklärungsaufgabe relevant sind, wie z.B. die Plattform selbst oder unterschiedliche Hardware, erweitert werden. Die erste Konzepterweiterung wurde von Hellert et al. in [42] vorgestellt.

4.1 Technische und funktionale Randbedingungen

Um die in den Kapitel 3 gesetzen Vorgaben an das Modell in Verbindung mit der Integration in das S&PM-Konzept erreichen zu können, gilt es eine Reihe von technischen und funktionalen Randbedingungen zu erläutern, welche zum Einen die Liste der Anforderungen an das Modell ergänzen bzw. zum Anderen eine Erweiterung der bestehenden S&PM-Architektur (siehe Kapitel 4.2) motivieren.

4.1.1 Bewertung auf Modulebene

Das in [39] eingeführte S&PM beschreibt Algorithmen und die daraus resultierenden perzeptiven Fähigkeiten in Bezug auf eine spezifische Aufklärungsaufgabe auf Kettenebene. Für jede Kette wird festgestellt, unter welchen Bedingungen sie verwendet werden kann und welche Einschränkungen für sie gelten. Die Konzepterweiterung sieht nun vor, diese Bewertung auf Modulebene durchzuführen (vgl. Kapitel 4.3). In Kapitel 2.2 wurde bereits gezeigt, dass zur Detektion von Objekten in Bilddaten verschiedene Schritte durchgeführt werden müssen, die zur redundanten Nutzung und besseren Wartung in Module gekapselt werden können. Da diese Module mehrfach und an verschiedenen Stellen des gesamten Algorithmus (der Perzeptionskette) eingesetzt werden können, muss eine von der Perzeptionskette getrennte und unabhängige Beschreibung eines Moduls existieren. Diese muss unter anderem Informationen darüber

enthalten, wie das Modul mit anderen Modulen zur Bildung einer Perzeptionskette verknüpft werden kann (vgl. Kapitel 2.3.2).

Die Bewertung auf Modulebene lässt sich wie folgt zusammenfassen:

- Der Wert einer Kette soll sich aus den Bewertungen ihrer Module berechnen lassen
- Die Beschreibung eines Moduls soll Informationen über den qualitativen Leistungsbeitrag enthalten (Wertung des Moduloutputs)
- Die Beschreibung eines Moduls muss dessen Vorbedingungen und Beschränkungen hinsichtlich der Aktionseigenschaften (z.B. „nur einsetzbar bis zu einer Flughöhe von 200 m“) enthalten
- Es müssen Bewertungskriterien vorliegen, die direkt den Algorithmus betreffen (z.B. False-Positives, Rechenzeit bzw. Verarbeitungsdauer, Speicherverbrauch, etc.)

Ziel ist es, funktionsredundante Ketten zu konstruieren, deren aktuelle Verwendbarkeit in Abhängigkeit von den gegebenen Umgebungsparametern und der jeweiligen UAV-Konfiguration bestimmt wird. Damit sollen die Anforderungen aus Kapitel 1.4 („Reaktionsfähigkeit bei Zustandsveränderungen“ und „Aussagefähigkeit hinsichtlich möglicher Ressourcenkombinationen“) erfüllt werden. Die Beschreibung der Module respektive Ketten aus dem Ressourcenmodell soll anderen Systemen zur Verfügung gestellt werden, damit diese die Informationen nutzen können, um mittels unterschiedlicher Metriken bestimmen zu können, in welcher Situation welche Kette genutzt werden soll [61] (siehe Kapitel 7.2).

4.1.2 Ressourcenerweiterung im Rahmen von Aufklärungsmissionen

Wie bereits zu Beginn dieses Kapitels angesprochen, berücksichtigt das S&PM lediglich bildgebende Sensoren an Bord eines unbemannten Luftfahrzeugs sowie Bildverarbeitungsalgorithmen. Obgleich diese beiden Ressourcenarten am Wichtigsten sind, zeigt die Analyse des technischen Systems zur Durchführung von Aufklärungsmissionen in Kapitel 2.1, dass weitaus mehr Komponenten (vgl. Kapitel 2.3.1) notwendig sind, um luftgestützte perzeptive Fähigkeiten zu realisieren. Daraus entsteht die folgende Anforderung:

- Alle Ressourcenarten (nach Kapitel 2.3) sollen in einem dedizierten Ressourcenmodell berücksichtigt werden.
- Es müssen geeignete Schnittstellen und Verarbeitungsroutinen innerhalb des S&PM zur Kommunikation und Verwaltung dieser Ressourcen bereitgestellt werden.

4.1.3 Integration des Ressourcenmodells

Um das Ressourcenmodell verwalten und nutzen zu können, soll das S&PM um eine Datenbank erweitert werden, in der semantische Beschreibungen von Fähigkeiten und Ressourcen gespeichert werden können. Diese Datenbank wird genutzt, um folgende Anforderungen erfüllen zu können:

- Bereitstellen von Schnittstellen zum Speichern, Lesen und Verändern des Ressourcenmodells
- Ermitteln von situationsabhängigen Ressourcenverknüpfungen und Systemparametern
- Reaktives Verhalten, Flexibilität und schnelle Zugriffszeiten in einem dynamischen Missionskontext
- Feststellen von Veränderungen (z.B. bei Systemausfällen)
- Ermitteln von Auswirkungen auf die Aufklärungsfähigkeiten im Fall von Zustandsveränderungen.

Weiterhin soll die Datenbank in Kombination mit geeigneten Verarbeitungsmechanismen verwendet werden, um

- alternative Lösungsstrategien für eine Aufgabe zu ermitteln (z.B. Anpassen einer Perzeptionskette durch den Austausch von redundanten Perzeptionsmodulen),
- das S&PM selbstständig an Situations-, Umgebungs-, und Systemveränderungen, ohne zwingende menschliche Intervention, anzupassen.

4.1.4 Kommunikation und Wartung

Zur Kommunikation des S&PM mit externen Systemen, wie z.B. dem *Mission Management System* (vgl. Abbildung 8) oder mit grafischen Werkzeugen zur Wartung und Überwachung ist die Definition von Kommunikationsschnittstellen notwendig.

- Die Schnittstellen sollen das Starten, Stoppen und Auswählen von Parametersets von Verarbeitungsketten erlauben.
- Aufklärungsergebnisse der jeweils ausgeführten perzeptiven Ketten sollen anderen Verarbeitungskomponenten zur Verfügung gestellt werden können.

Das S&PM führt zwar perzeptive Ketten aus, jedoch findet keine Überwachung dieser Ketten statt. Der aktuelle Zustand (welche Ketten werden gerade ausgeführt; aus welchen Modulen besteht eine Kette; welche Module prozessieren und welche Module warten auf Daten; etc.) der Ketten ist nicht bekannt. Auch findet keine Wiederverwendung von Modulen statt. Die Module kapseln bestimmte bildverarbeitende Funktionalitäten, sodass sie in verschiedenen Ketten wiederverwendet werden können. Werden jedoch mehrere Ketten mit identischen Modulen gestartet, werden diese Module mehrfach gestartet. Als Forderungen ergeben sich daher:

- Module sollen in verschiedenen Ketten wiederverwendet werden, ohne dass diese mehrfach ausgeführt werden müssen.
- Zu Wartungs- bzw. Visualisierungszwecken sollen Schnittstellen bereitgestellt werden, die es erlauben, den Zustand der Datenbank, die vorhandenen Perzeptionsketten samt der beanspruchten Ressourcen und den Gesamtzustand des UAVs auf einem externen Gerät (z.B. an der Bodenkontrollstation) auszulesen.

4.1.5 Kompatibilität zwischen Realflug und Simulation

Das S&PM wurde unter dem Gesichtspunkt weiterentwickelt, in verschiedenen Szenarien mit unterschiedlichen Konfigurationen eingesetzt werden zu können. Zu diesen zählen sowohl Missionen in der Simulation als auch im Realflug im Rahmen von Einzel-UAV als auch Multi-UAV-Missionen. Aus diesem Zusammenhang resultieren die folgenden Anforderungen:

- Je nach Ausstattung der UAVs (Sensoren, Algorithmen, Gimbal, etc.) soll sich das S&PM weitgehend selbst konfigurieren, die zur Verfügung stehenden Funktionen bzw. Fähigkeiten ermitteln und diese bereitstellen.
- Das S&PM soll soweit modular aufgebaut sein, dass für einen funktionellen Einsatz nicht alle Komponenten bzw. Softwaresysteme bereitgestellt werden müssen. Der modulare Aufbau hilft zudem, das System robuster gegen Systemausfälle und Softwarefehler zu schützen.
- Ressource-Komponenten sollen, soweit möglich, zur Laufzeit aktualisiert bzw. ausgetauscht werden können.
- Das S&PM soll die unterschiedlichen Hard- und Softwareschnittstellen und Protokolle in Simulation und Real-Flug abstrahieren und kapseln, um so die Einsetzbarkeit und Übertragbarkeit zu erleichtern.

4.2 Aufbau und Funktionsweise des erweiterten S&PM

Abbildung 21 zeigt die Schichtenarchitektur des erweiterten S&PM. Um die Anforderung der Integrierbarkeit in das bestehende Systemkonzept (vgl. Kapitel 1.4) zu erfüllen, wurde die Systemkonfiguration zum größten Teil beibehalten und nur an notwendigen Stellen verändert bzw. erweitert. Nach wie vor bildet die oberste Schicht das *Perception Management*. Anstelle der *Sensor Management* Schicht (vgl. Abbildung 8) findet man nun darunterliegend das *Resource Management*, welches wiederum auf die Ressourcen zugreift. Während das von Russ eingeführte Sensor Management des S&PM ausschließlich für die Sensorik zuständig war, enthält die Erweiterung Subprozesse in der *Resource Management* Schicht, um die verschiedenen Ressourcentypen zu verwalten. Das übergeordnete *Mission Management* selbst ist nicht Teil des S&PM.

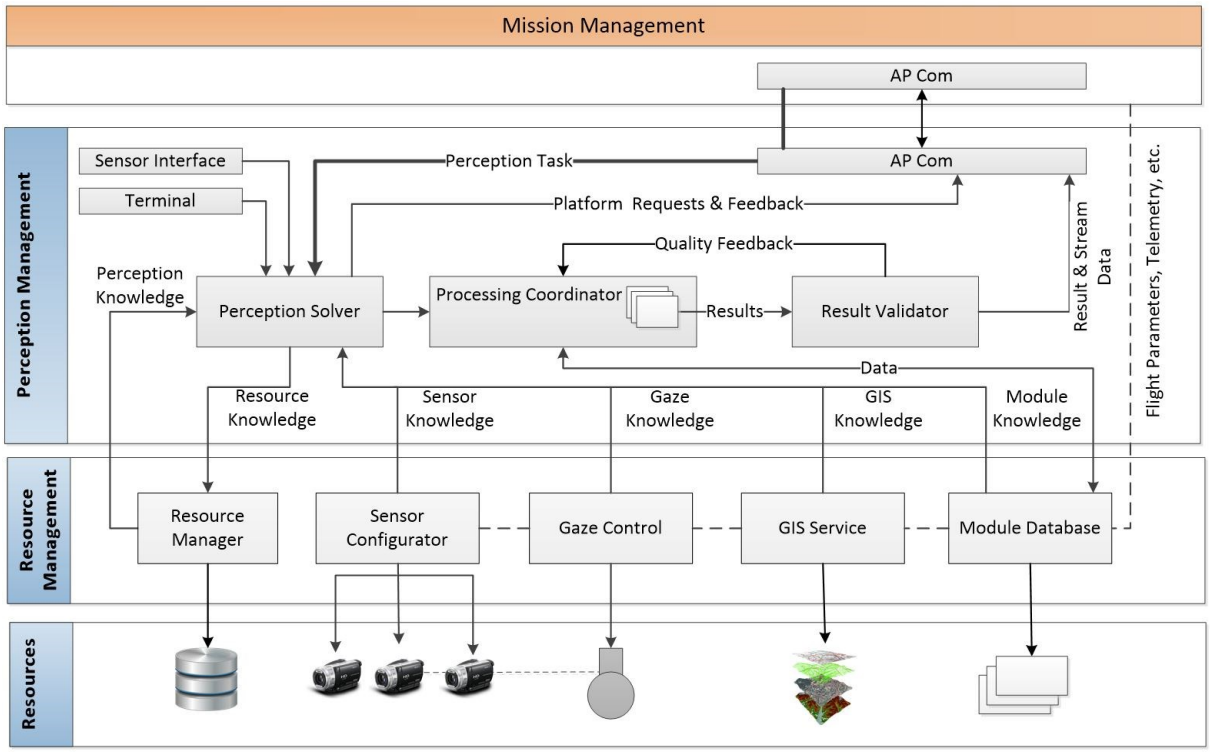


Abbildung 21: Konzepterweiterung des Sensor- und Perzeptionsmanagementsystems

Die wichtigste Veränderung, um die Anforderungen aus Kapitel 1.4 zu erfüllen, ist die Einführung einer Ressourcendatenbank und dem damit verbundenen Verwaltungsprozess. In der untersten Schicht ist neben den verschiedenen Ressourcen (vgl. Kapitel2.3.1) die Ressourcendatenbank für das Ressourcenmodell zusehen. Die Ressourcen sind mit Managementprozessen, welche in der Ebene des *Resource Managements* angeordnet sind, verknüpft. Eine Sonderrolle nimmt dabei der *Resource Manager* ein. Dieser bildet die Kommunikationsschnittstelle zwischen der Datenbank und dem *Perception Solver*, welcher wiederum den zentralen Prozess im Gesamtsystem abbildet. Im Folgenden werden die einzelnen Komponenten näher erläutert.

Resource Manager:

Der *Resource Manager* ist für die Verwaltung des Ressourcenmodells zuständig. Dafür besitzt er eine Schnittstelle zur Datenbank, um Informationen zu speichern, laden und verändern zu können. Außerdem tauscht er Informationen (*Perception Knowledge* und *Resource Knowledge*) mit dem *Perception Solver* aus. Vom *Solver* werden Informationen über den aktuellen Zustand (welche Ressourcen sind vorhanden, welche Aufgabe soll durchgeführt werden, etc.) der Ressourcen an den *Manager* übertragen. Mit dem in der Datenbank gespeicherten Ressourcenmodell und den vom *Solver* übermittelten Informationen kann der *Resource Manager* bestimmen, ob bzw. wie eine Perzeptionsaufgabe durchgeführt werden kann und wie Ressourcen kombiniert werden müssen, damit eine bestimmte Fähigkeit angeboten werden kann. Die Implementierung des *Resource Manager* ist darauf ausgelegt, die Anforderung aus Kapitel 1.4 an Aussagefähigkeit und Reaktionsfähigkeit zu erfüllen. Damit auch die Anforderung an eine begrenzte Verarbeitungszeit eingehalten wird, wird durch die Wahl einer effizienten Datenbank ein performanter Datenaustausch gewährleistet.

Sensor Configurator:

Der *Sensor Configurator* stellt Schnittstellen zur Kommunikation mit den verschiedenen Sensoren bereit. Die Schnittstellen bedienen unterschiedliche Protokolle (GigE, USB, Cameralink, virtuelle Ethernet-Sensoren). Der Prozess erkennt selbstständig, sobald Sensoren angeschlossen bzw. getrennt werden und kommuniziert Bildinformationen und Metadaten[13] der Sensoren an den *Perception Solver*. Außerdem wird der *Sensor Configurator* genutzt, um die Sensoren zu parametrisieren bzw. zu konfigurieren. Im *Perception Solver* können Kontrollfunktionen verwendet werden, die dem *Sensor Configurator* Befehle übermitteln, um Sensoren, sofern diese in einer ausgewählten Kette benötigt werden, zu aktivieren bzw. zu deaktivieren.

Gaze Control:

Gaze Control ist für die Steuerung einer oder mehrerer kardanischer Sensoraufhängungen (engl. Gimbal) zuständig. Weiterhin können im *Gaze Control* höherwertige Modi wie z.B. der Geolock[14] oder Stabilisierungsfunktionen an den Gimbal kommandiert werden. Ebenso wie der *Sensor Configurator* überträgt die *Gaze Control* ebenfalls Metainformationen wie z.B. Funktionsumfang und Achsenlimitationen der angeschlossenen Gimbals an den *Perception Solver.*

GIS Service:

Um das Spektrum an möglichen Aufklärungsaufgaben präziser bedienen zu können, wurde ein Geoinformationssystem (GIS) integriert, welches aus zwei Komponenten besteht: Zum einen aus der GIS Datenbank selbst, die nach dem Konzept in Abbildung 21 als Ressource interpretiert wird. Die Datenbank enthält georeferenziertes Kartenmaterial zum Einsatzgebiet. Zum anderen besteht das GIS aus der Softwarekomponente, welche erforderlich ist, um Anfragen an die GIS Datenbank stellen zu können. Die Informationen können verwendet werden, um z.B. Ergebnisse (Objektdetektionen) mit georeferenzierten Koordinaten zu verknüpfen, die Geokoordinaten der Eckpunkte des Sensorfootprints[15] (*Latitude*, *Longitude* und *Altitude*) zu berechnen oder um die Koordinaten von topographisch relevanten Bereichen im Missionsgebiet zu bestimmen.

Module Database:

Die *Module Database* enthält die Beschreibung (IDs, Funktionsumfang, Beschränkungen, etc.) aller bildverarbeitenden Algorithmen (Module), jedoch nicht die ausführbaren Algorithmen selbst (vgl. Kapitel 4.3). Sie wird vom *Processing Coordinator* genutzt, um vorhandene Bildverarbeitungsmodule starten, stoppen und verwalten zu können.

[13] Herstellerinformationen, Seriennummer, IP-Adresse, Parameter, etc.
[14] Das Fixieren einer Geokoordinate am Boden, sodass sich der Bildausschnitt nicht ändert, unabhängig von der Bewegung des Luftfahrzeugs.
[15] Sensorbildmarkierung in Form eines Trapezes auf der Karte, der sich mit dem FOV (*Field of View*) des Sensorbilds deckt

Das Ressourcenwissen (welche Sensoren und Module sind vorhanden, welche Funktionen bietet der Gimbal, welche Services sind angebunden, etc.) wird von den jeweiligen Managementprozessen an den *Perception Solver* übermittelt. Dieser wiederum leitet diese Informationen (*Ressource Knowledge*) an den *Resource Manager* weiter.

Perception Solver:

Der *Perception Solver* nimmt eine zentrale Rolle im System ein. Im *Solver* laufen alle Informationen zusammen und werden von dort aus an die angeschlossenen Komponenten weiter verteilt. Der *Solver* fungiert zudem als Kommunikationsserver (*AP Com*), an den z.B. die perzeptiven Aufgaben (*Perception Task*) vom *Mission Management* System (MMS) übermittelt werden. Gleichzeitig ist vorgesehen, dass der *Solver* Anfragen über das MMS an die fliegende Plattform senden kann, um z.B. die Flughöhe zu reduzieren oder den Betrachtungswinkel auf ein bestimmtes Ziel zu verändern. Zusätzlich sendet der *Solver* eine Liste der vorhandenen perzeptiven Fähigkeiten (*Platform Requests & Feedback*) an das übergeordnete MMS. Neben der Kommunikation über die *AP Com* mit externen Systemen, besitzt der *Solver* Schnittstellen, über die direkt mit dem *Solver* über z.B. ein Terminal oder eine GUI kommuniziert bzw. die Aktivität des *Solvers* überprüft und beobachtet werden kann (vgl. Anforderung aus Kapitel 4.1.4).

Der *Solver* speichert Informationen, welche Ressourcen vorhanden sind und welche perzeptiven Ketten ausgeführt werden. Findet eine Änderung im System statt (z.B. Sensor- oder Hardwareausfall), wird der *Solver* darüber informiert, woraufhin dieser die Informationen an die betroffenen Subsysteme (z.B. im Falle eines Sensorausfalls an den *Resource Manager*, damit dieser den Sensor in der Datenbank deaktivieren und die neuen Fähigkeiten ermitteln kann) weiterleitet und gegebenenfalls notwendige Maßnahmen, wie z.B. das Neukonfigurieren einer Kette oder Starten einer redundanten Kette mit derselben Fähigkeit, einleitet. Um eine Kette zu starten, erhält der *Solver* über die *AP Com* Informationen darüber, welche perzeptive Fähigkeit bzw. Fähigkeiten für eine bestimmte Aufgabe benötigt werden. Zusätzlich zu der zu startenden Fähigkeit übermittelt die *AP Com* auch angepasste Parametersets, mit denen die Module gestartet werden sollen. Diese Sets können algorithmenspezifische Parameter enthalten, aber auch Informationen, wie z.B. der Sensor konfiguriert sein soll. Der *Solver* gleicht die empfangenen Daten mit den in der Ressourcendatenbank abgelegten Fähigkeiten und den entsprechend vorhandenen Ketten ab, wählt eine passende Kette aus, passt, sofern notwendig, die mitübermittelten Parameter an und sendet die Informationen (ID, Parameter, etc.) für die zu startenden Module an den *Processing Coordinator*.

Processing Coordinator

Der *Processing Coordinator* kontrolliert das Starten, Stoppen und Verwalten von Perzeptionsmodulen. Er sorgt dafür, dass Module, die in derselben Konfiguration von mehreren Ketten verwendet werden, diesen zugeordnet werden und nicht mehrfach gestartet werden müssen. Sendet der *Solver* einen Befehl an den *Coordinator* zum Starten einer Perzeptionskette, erwidert

dieser den Befehl, je nachdem ob die Kette gestartet werden konnte oder nicht, mit einem positiven oder negativen Feedback. Da der *Solver* und der *Coordinator* als unabhängige Prozesse realisiert wurden, die ihre Daten über eine Interprozesskommunikation austauschen, können sie sich nach einem Programmabsturz wieder synchronisieren. Diese softwareseitige Redundanz dient der Systemstabilität. Die vorhandenen Module werden dem *Processing Coordinator* über die *Module Database* mitgeteilt. Die von den aktivierten *Modulen* produzierten Perzeptionsergebnisse werden an den *Result Validator* übermittelt.

Result Validator

Der *Result Validator* übernimmt zwei Funktionen: Zum einen dient er als Ergebnisdatenbank, sodass auf die Ergebnisse zu unterschiedlichen Zeiten und von verschiedenen Prozessen zugegriffen werden kann. Die Ergebnisse werden dabei nach *Perception Tasks* sortiert, sodass sich z.B. Ergebnisse zu einer bestimmten Aufgabe ermitteln lassen. Zum anderen wird der *Validator* genutzt, um die Ergebnisse qualitativ zu prüfen. Der *Validator* verfügt über verschiedene Metriken und Heuristiken, mit denen Ergebnisse beurteilt werden können. Der Zugriff auf den *Result Validator* findet über die *AP Com* statt.

4.3 Perzeptionsmodule, -Ketten und –Graphen

Unter den in Kapitel 2.3.1 vorgestellten Ressourcen spielen die Datenverarbeitungsalgorithmen (Perzeptionsmodule) eine wichtige Rolle im S&PM. Durch die Module ist es möglich, Bildverarbeitung und damit z.B. Objektdetektion und Tracking durchzuführen (vgl. Kapitel 2.2) [43]. Im Folgenden sollen der Unterschied und das Zusammenspiel zwischen Perzeptionsmodulen, -ketten und dem Graphen erläutert werden.

4.3.1 Perzeptionsmodule

Perzeptionsmodule kapseln Algorithmen (im *Processing Core,* Abbildung 22), die für sich abgeschlossen eine bestimmte Datenverarbeitung durchführen (vgl. Kapitel 2.2). Aus mehreren Modulen entsteht eine Perzeptionskette. Die Komplexität eines Moduls hängt von den Verarbeitungsschritten ab. Werden diese mehrfach an unterschiedlichen Stellen benutzt, ist es ratsam, sie in Module zu kapseln. Werden bestimmte Schritte immer in derselben Reihenfolge ausgeführt und sind diese somit voneinander abhängig, ist es sinnvoll, diese Schrittabfolge in einem einzigen Modul zu kapseln, statt jeden Schritt in einem eigenen Modul abzulegen. In Abbildung 22 ist das Konzept eines Perzeptionsmoduls und die Verknüpfung mit dessen Ressourcenbeschreibung dargestellt.

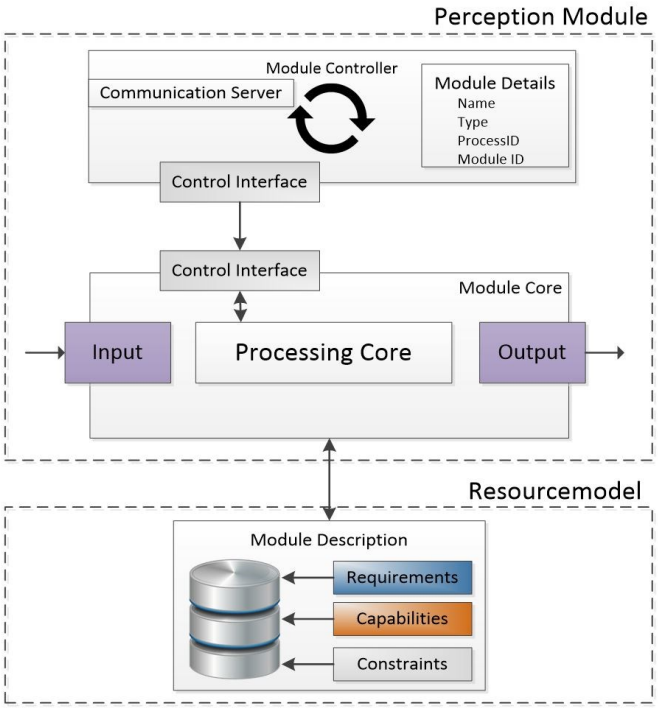


Abbildung 22: Konzeptionelle Darstellung eines Perzeptionsmoduls und dessen Ressourcenbeschreibung

Weitere Bestandteile des *Processing Core* sind die *Input*-, *Output*- und die Kontroll-Schnittstelle (*Control Interface*). Die *Input*- und *Output*- Schnittstelle wird genutzt, um sicherzustellen, dass die richtigen Daten in einem für das jeweilige Modul konformen Datenformat ankommen. Die Kontrollschnittstelle ermöglicht es, das Modul über den *Module Controller* zu steuern und Details wie z.B. den aktuellen Status, Name oder die ID des Moduls auszulesen. Der *Processing Coordinator* aus Abbildung 21 kann sich über den *Communication Server* mit dem *Module Controller* verbinden und das Modul starten, stoppen oder die Konfigurationsparameter während des Betriebs verändern.

Jedes Modul besitzt eine eigene Modulbeschreibung (*Module Description*), die im Ressourcenmodell hinterlegt ist. Diese Beschreibung enthält folgende Informationen:

- Vorbedingungen bzw. Anforderungen (*Requirements*) an die Inputdaten: Zum Beispiel Datenformat (RGB, Y08, etc.), Auflösung, Bildsignatur (Infrarot oder EO Bilddaten), etc.
- Modulspezifische Parameter bzw. Parametersets: Klustergröße, ROI-Anzahl, Featurelisten aus Trainingsdaten, Filter, etc.
- Fähigkeit(en) (*Capabilities*): Was leistet das Modul und wie bzw. in welchem Format stellt es die Daten zur Verfügung (z.B. detektiert Fahrzeuge in Bilddaten und stellt diese in Form von ROIs bereit).
- Restriktionen (*Constraints*): In welcher Situation ist das Modul nicht mehr in der Lage, verwertbare Ergebnisse zu produzieren. Zu den Restriktionen gehören flugspezifische Randbedingungen wie z.B. (minimale / maximale) Geschwindigkeit und Flughöhe, aber auch Prozess- bzw. missionsbedingte Eigenschaften wie z.B. Zeit-, Wetter-, Beleuchtungs-, Sensor- und rechenbezogene Restriktionen.

Wie in Kapitel 2.2 beschrieben, werden die Schritte eines Bildverarbeitungsalgorithmus in verschiedene Gruppen unterteilt. Entsprechend wurden die Module ebenfalls in funktionelle Gruppen eingeteilt. Diese Vorgehensweise wird genutzt, um wiederkehrende Verarbeitungsschritte in komplexen Algorithmen zu implementieren und in unterschiedlichen Ketten wieder zu verwenden. Die Gruppierung soll helfen, Bildverarbeitungsfunktionen nach einem Baukastenprinzip bereitzustellen. Sind innerhalb einer Kategorie mehrere Module vorhanden, können diese untereinander (je nach Hardwareimplementierung und Qualität der Ergebnisse) getauscht werden. Wird ein neues Modul einer Kategorie hinzugefügt, können bereits existierende Perzeptionsketten derart verändert werden, dass das alte Modul aus der Kategorie durch die neue Implementierung ersetzt wird und dadurch die Qualität der ganzen Kette eventuell gesteigert werden kann. Fällt eine Hardwareressource (z.B. GPU) aus, kann ein Modul aus derselben Kategorie, jedoch einer anderen Implementierung (z.B. CPU), den Ausfall kompensieren, sodass die Perzeptionskette weiterhin ausgeführt werden kann. Abbildung 23 zeigt die Anordnung der verschiedenen Kategorien und wie Module (Algorithmen) miteinander Kategorie übergreifend verknüpft werden können.

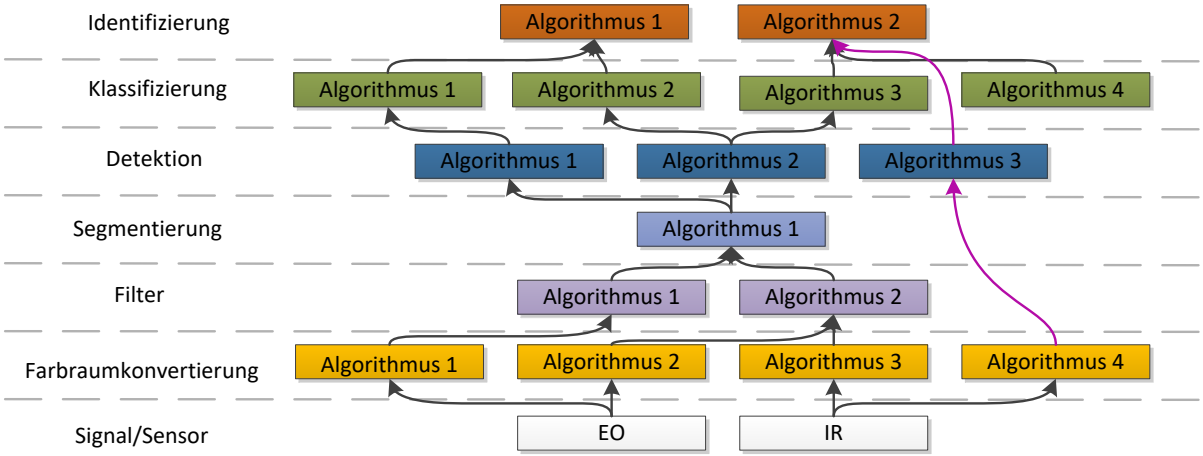


Abbildung 23: Schemenhafte Darstellung der Gruppierung von verschiedenen Bildverarbeitungsmodulen. In der untersten Ebene sind die Sensoren abgebildet, die die Module mit Sensordaten versorgen.

In den meisten Fällen ist ein Modul mit einem anderen Modul aus der Ebene darüber verbunden (vgl. Kapitel 2.2). Es gibt jedoch auch Fälle, in denen ein Modul mehrere Gruppenebnen überspringt (vergleiche Abbildung 23 lila Pfeile), da die dazwischenliegenden Verarbeitungsschritte nicht benötigt werden.

4.3.2 Perzeptionskette

Eine Perzeptionskette bildet sich aus der Konkatenation mehrerer Perzeptionsmodule. Während ein Modul Teilaufgaben löst, stellt eine Perzeptionskette eine Lösung für ein komplexes perzeptorisches Problem bereit. Eine Kette gruppiert und sortiert eine Abfolge von Verarbeitungsschritten (in Form von Modulen), die durchgeführt werden müssen, um bestimmte Informationen aus Daten extrahieren zu können. Die Gesamtfähigkeit einer Kette ergibt sich aus der Beschreibung der kombinierten Modulfähigkeiten. In Abbildung 13 des Kapitels 2 ist die Idee einer Kette dargestellt. Stehen Module bereit, die von ihrer Fähigkeit her identisch sind, sich

jedoch in ihrer Implementierung unterscheiden, entstehen redundante Ketten, die dasselbe Problem lösen, sich jedoch in ihrer Modulkonfiguration unterscheiden. Dabei kann das Ergebnis einer Kette sowohl in Qualität als auch Quantität variieren.

Je nach Situation und Anforderung muss eine geeignete Kettenkonfiguration gewählt werden. In bestimmten Fällen sind schlechtere Ergebnisse akzeptabel, wenn dadurch die Verarbeitungsdauer reduziert werden kann oder weniger Ressourcen notwendig sind. In anderen Fällen liegt die Priorität auf einer möglichst hohen Qualität der Ergebnisse, auch wenn dadurch aufgrund hohen Rechenaufwands eine Verzögerung der Bereitstellung von Ergebnissen eintritt.

4.3.3 Perzeptionsgraph

Der Perzeptionsgraph bildet sich aus der Summe aller Perzeptionsketten bzw. aller sinnvollen Permutationsmöglichkeiten der Module.

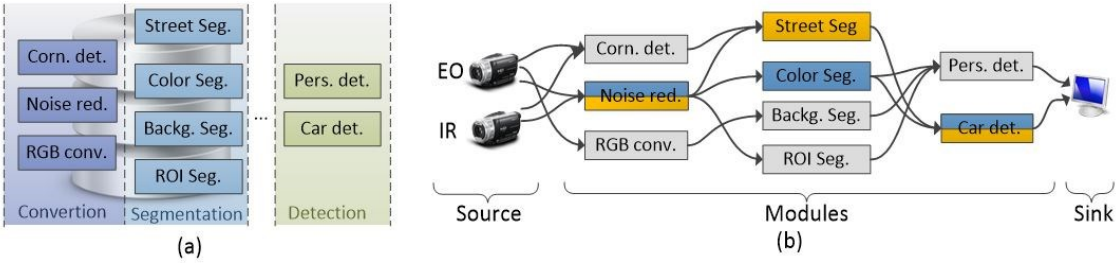


Abbildung 24: Perzeptionsgraph. (a) Module in einer Datenbank, sortiert nach Kategorien. (b) Perzeptionsgraph.

Abbildung 24 zeigt die Darstellung eins solchen Graphen. In (a) ist ein Ausschnitt aus der Datenbank mit Perzeptionsmodulen, sortiert nach verschiedenen Kategorien, zu sehen (vgl. Abbildung 23). Aus diesen Modulen kann in (b) ein gerichteter Graph zusammengesetzt werden. Die farblich markierten Module in (b) repräsentieren zwei Perzeptionsketten. Die blau-gelben Module zu Beginn und Ende der Kette kommen in beiden Ketten zum Einsatz. Das gelbe bzw. blaue Modul wird nur in der jeweiligen Kette genutzt. Die grauen Module sind Bestandteil anderer Ketten, die nicht zur benötigten Fähigkeit *Fahrzeugerkennung* beitragen.

4.4 Fähigkeiten und Ressourcen-Out/Inputs

Aus Kapitel 2.3.2 ist bekannt, dass für eine Ressource zwischen drei verschiedenen Beschreibungen unterschieden wird:

(1) $R_{Capability}$,
(2) R_{Input} und
(3) R_{Output}

(1) wird verwendet, um zu ermitteln, welche Ressourcen für eine Perzeptionsaufgabe in Frage kommen. (2) und (3) werden für die Vor- und Nachbedingung bei der Verknüpfung von Ressourcen benötigt. Es müssen nicht alle drei Beschreibungen für jede Ressource vorhanden sein.

Ein Sensor z.B. besitzt nur eine Beschreibung des Outputs und der Fähigkeit (z.B. thermische Signaturen detektieren). Für einen Gimbal existiert keine Beschreibung von Input- bzw. Output, sondern nur eine Beschreibung seiner Fähigkeiten (z.B. Sensorstabilisierung).

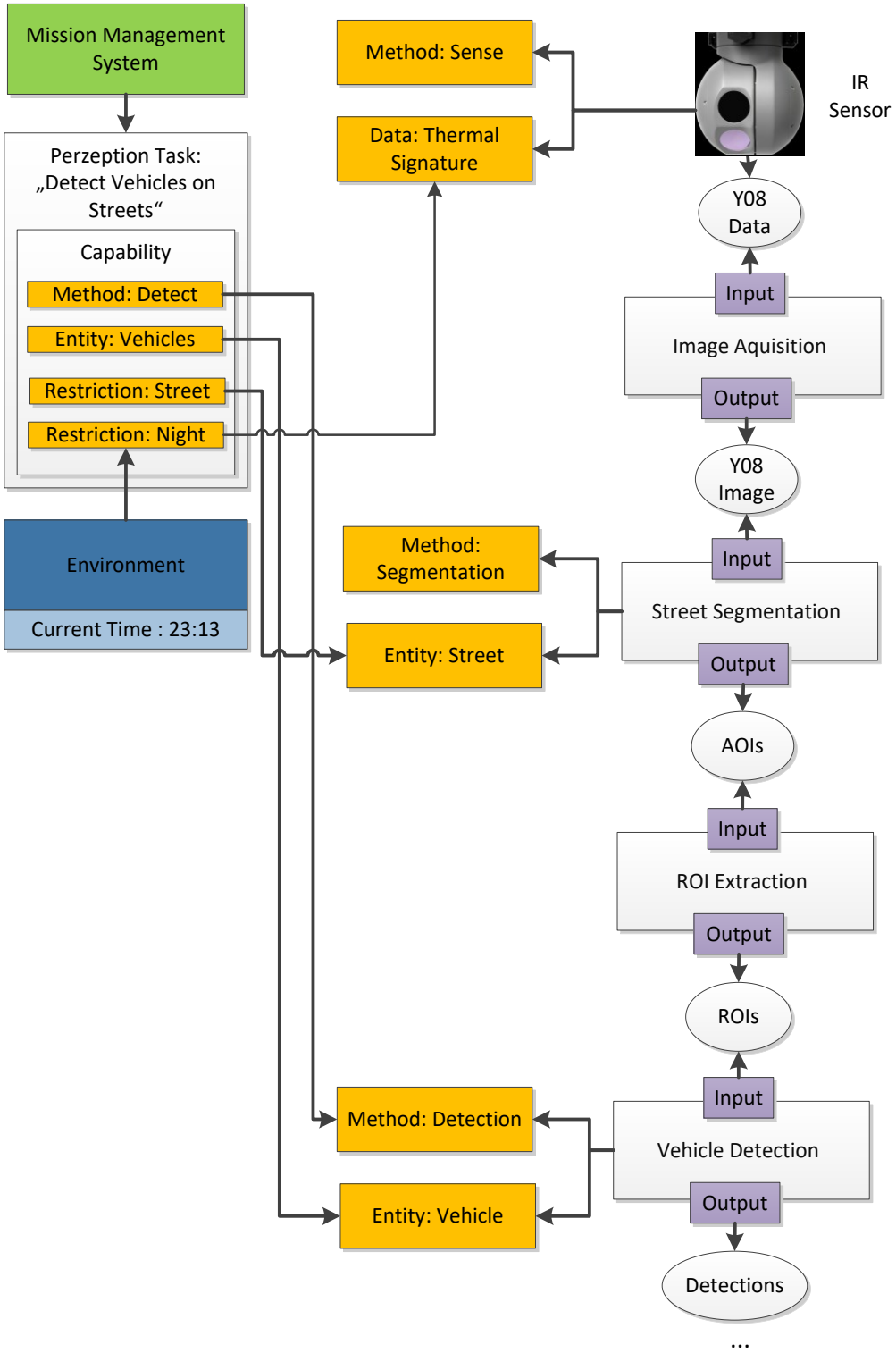


Abbildung 25: Beispielhafte Perzeptionskette mit unterschiedlichen Fähigkeiten und Module-In/Outputs

In Abbildung 25 wird anhand eines Beispiels der Unterschied zwischen der Fähigkeitenbeschreibung (1) und dem In- (2) bzw. Output (3) verdeutlicht.

Die linke Seite zeigt die Beschreibung einer Gesamtfähigkeit, wie sie vom *Mission Management System* angefordert wird. In diesem Beispiel sollen Fahrzeuge auf einer Straße detektiert werden. Zusätzlich wird vom System automatisch aufgrund der aktuell angenommen Missionszeit von 23:13 Uhr (*Environment*) implizit die Restriktion „Nacht“ ermittelt, sodass eine Perzeptionskette gefunden werden muss, die die vom *Mission Management System* geforderte Aufgabe bei Nacht erfüllen kann. Die technische Beschreibung dieser Mission, wie sie an das System übermittelt wird, setzt sich aus der Methode („detektieren“), der gesuchten Entität („Fahrzeug“) und zwei Restriktionen („Straße“, „Nacht“) zusammen, dargestellt durch die orangenen Rechtecke. Diese Kombination aus Fähigkeiten muss mit der Kombination von $R_{Capability}$ Beschreibungen von Ressourcen übereinstimmen, damit eine gültige Perzeptionskette ermittelt werden kann.

Auf der rechten Seite der Abbildung ist eine Perzeptionskette inklusive eines Infrarot Sensors dargestellt, die die geforderte Aufgabe unter den herrschenden Bedingungen erfüllen kann. Der Infrarotsensor und zwei Bildverarbeitungsmodule besitzen eine Beschreibung $R_{Capability}$ (orange Rechtecke), über die ermittelt werden kann, dass durch eine Kombination dieser Ressourcen die geforderte Aufgabe erfüllt werden kann. Die Beschreibung enthält sowohl eine Angabe über die Methode als auch über die Entität. Wie bereits erwähnt, muss nicht jede Ressource eine Beschreibung $R_{Capability}$ besitzen. Um eine gültige Kombination der notwendigen Ressourcen zu ermitteln (Vor- und Nachbedingungen müssen erfüllt sein), wird die Beschreibung R_{Input} und R_{Output} der Ressourcen betrachtet (Lila Rechtecke, bzw. weiße Ellipsen). Durch ein Mapping der Input- und Outputbeschreibung der Ressourcen kann eine gültige Ressourcenkombination ermittelt werden.

4.4.1 Die Beschreibung $R_{Capability}$

Die Beschreibung einer Fähigkeit ($R_{Capability}$) setzt sich aus der angewandten Methode (z.B. detektieren) und der Entität (z.B. Fahrzeug), auf die die Methode angewendet werden soll, zusammen. Neben einer Entität kann eine Fähigkeit auch lediglich Daten beschreiben (z.B. Bildformat: JPG). Dies ist dann der Fall, wenn Module z.B. nur ein Video von einem bestimmten Gebiet zur Verfügung stellen, ohne die Daten zu manipulieren, d.h., keine konkreten Bildoperationen auf den Daten ausführen.

Zusätzlich können Beschränkungen der Fähigkeitenbeschreibung hinzugefügt werden. Solche Beschränkungen können nähere Umstände erläutern (z.B. Tageszeit oder Ort), unter denen eine Fähigkeit eingesetzt werden soll. Damit lässt sich eine Fähigkeit wie folgt beschreiben:

$$Fähigkeit = \left\{ \begin{matrix} Methode \\ Entität|Daten \\ \sum Beschränkungen \end{matrix} \right\}$$

Im Folgenden soll anhand eines Beispiels der Inhalt und Aufbau einer Fähigkeitenbeschreibung genauer erläutert werden. Die aufgezählten Methoden und Entitäten sind nicht vollständig und dienen nur der Veranschaulichung. Die implementierten Methoden und Entitäten sind in Kapitel 5.3.3 erläutert. Folgende Methoden für eine luftgestützte Aufklärungsmission stehen zur Verfügung:

$$Methode = \{Detect, Identify, Record, Stream, Track, Sense, Segment, \dots\}$$

Die Entitäten werden in drei Gruppen unterteilt:

$$Entität = \{\, Human, Structure, Vehicle\}$$

Jede der Gruppen enthält näher spezifizierte Begriffe zur genaueren Kategorisierung:

$$Human = \{Human, Civilian, Soldier, \dots\}$$

$$Structure = \{Structure, Airport, Church, Hospital, Fabric, Street, \dots\}$$

$$Vehicle = \{Vehicle, Bus, Car, Tank, Truck, Motorcycle, \dots\}$$

Die folgenden Datenbeschreibungen stehen zur Verfügung:

$$Daten = \{Image, Video, Result, Thermal\ Energy, Visible\ Light,\}$$

Schließlich kann die Beschreibung einer Fähigkeit verschiedene Beschränkungen beinhalten. Beispiele hierfür sind:

$$Beschränkung = \{Daytime, Location, \dots\}$$

$$Daytime = \{Dawn, Dusk, Day, Night|\ Time\}$$

$$Location = \{Street, City, Forest, Sea, \dots \mid Coordinates \mid Area\ \}$$

Zwei gültige Fähigkeitenbeschreibungen sind im Folgenden dargestellt:

$$Fähigkeit_X = \begin{Bmatrix} Identify \\ Truck \\ \sum N/A \end{Bmatrix} \qquad Fähigkeit_y = \begin{Bmatrix} Stream \\ Video \\ \sum Night \end{Bmatrix}$$

Die Fähigkeit X beschreibt das Identifizieren des Fahrzeugtyps *Truck*. Es werden keine Informationen über Einschränkungen gemacht. Fähigkeit Y beschreibt die Möglichkeit, Videodaten zu *streamen*[16]. Als Restriktion ist definiert, dass das Streamen dieser Daten bei Nacht durchgeführt werden soll.

[16] Das Übertragen von Daten in Form kleiner Paketgrößen

Aus der Auswahlmöglichkeit zwischen Entitäten und Daten entstehen Restriktionen für die Wahl der Methoden. Nicht jede Methode kann mit jeder Entität verknüpft werden. Deshalb müssen die Methoden noch einmal in separate Kategorien getrennt und ein Prüfmechanismus bereitgestellt werden, der darauf achtet, dass nur sinnvolle Kombinationen in Frage kommen. Weiterhin muss die Möglichkeit bestehen, dass bestimmte Methoden gleichzeitig zu beiden Kategorien gehören können (vgl. Abbildung 26).

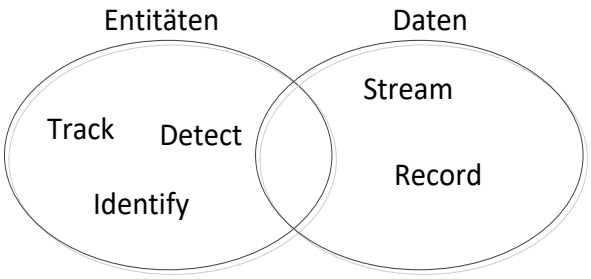


Abbildung 26: Individuen der Klasse "Methode", getrennt in zwei Subklassen

Es ist also möglich, z.B. ein Fahrzeug zu detektieren, identifizieren und zu verfolgen. Jedoch ist es nicht möglich, das Fahrzeug zu streamen bzw. aufzuzeichnen. Diese Methoden sind ausschließlich mit *Image*, *Video* oder *Result* kombinierbar. Umgekehrt ist es nicht möglich, z.B. ein Ergebnis (*Result*) zu detektieren oder zu verfolgen.

5

Repräsentationsformen für das Ressourcenmodell

Um die in den vorhergehenden Kapiteln erläuterten Ressourcen sowie deren Fähigkeiten samt Vor- und Nachbedingungen in einem Computersystem nutzen zu können, müssen sie mittels eines Modells repräsentiert werden. Hierfür stehen verschiedene Methoden und Formalismen zur Verfügung.

Damit eine Maschine Informationen interpretieren und verarbeiten kann, müssen diese formal vorliegen. Entsprechende Formalismen werden auch als *Wissensrepräsentation* bezeichnet. Softwarearchitekturen, die explizit modellierte Wissenszusammenhänge nutzen und damit u.a. eine Designgrundlage für automatisierte intelligente Systeme bilden, werden *wissensbasierte Systeme* genannt [64] - [68] [71].

Die generische Architektur (vgl. Abbildung 27) eines solchen wissensbasierten Systems [64] umfasst eine Reihe von Komponenten, die notwendig sind, um explizites Wissen zu verarbeiten und anderen Entitäten zur Verfügung stellen zu können. Der Kern besteht aus der bereits erwähnten *Wissensbasis*. Die *Wissensbasis* ist verknüpft mit der Problemlösungskomponente. Ziel dieser Komponente ist es, zielgerichtet (neue) Erkenntnisse in der *Wissensbasis* herzustellen. Das Erschließen neuer Erkenntnisse (schlussfolgern*; engl. reasoning*) aus der formalen Beschreibung wird als *Inferenz* bezeichnet und in Form von *Inferenzmechanismen* umgesetzt.

Neben diesen Hauptkomponenten erlaubt eine (optionale) *Erklärungskomponente*, die Entstehung einer konkreten Schlussfolgerung zu hinterfragen. Eine geeignete Schnittstelle (im Beispiel als *Benutzeroberfläche* beschrieben) erlaubt die Formulierung der zu lösenden Fragestellung sowie der Entgegennahme von Resultaten, die über die Wissensakquisitionskomponente bereitgestellt werden.

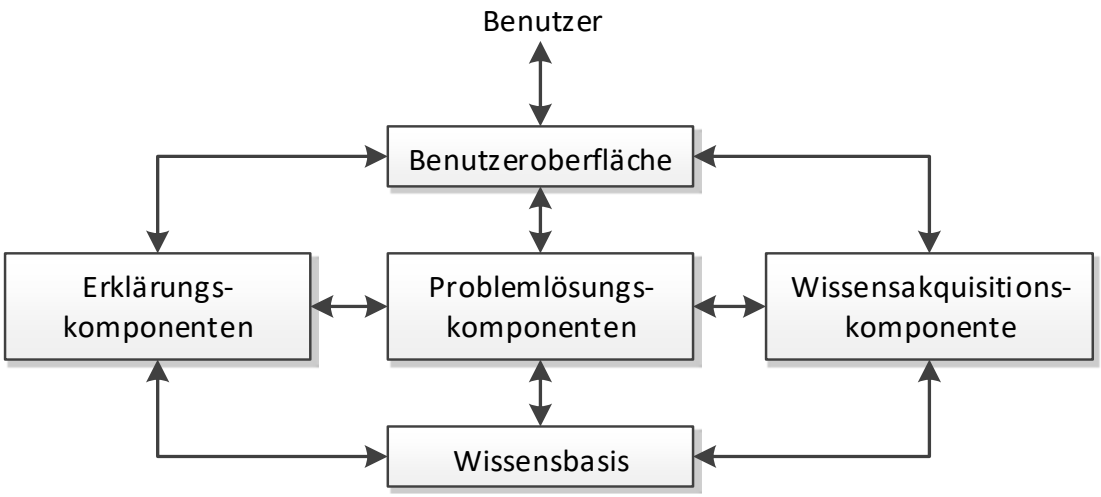


Abbildung 27: Allgemeine Architektur wissensbasierter Systeme [67]

Die Notwendigkeit einer formalen Beschreibung, damit eine Maschine einen Sachstand interpretieren kann, soll am folgenden Beispiel veranschaulicht werden (vgl. Abbildung 25). Es sei angenommen, dass folgende Fakten existieren:

- Es steht ein mit zwei Sensoren ausgestattetes unbemanntes Luftfahrzeug zur Verfügung. Bei den Sensoren handelt es sich um einen elektrooptischen und einen Infrarotsensor. Sensoren produzieren Sensordaten.
- Weiterhin besitzt das UAV einen Computer. Auf dem Computer können Bildverarbeitungsalgorithmen ausgeführt werden.
- Mittels Bildverarbeitungsalgorithmen können Bilddaten auf Fahrzeughypothesen hin überprüft werden. Bildverarbeitungsalgorithmen benötigen Sensordaten.

Weiterhin sei angenommen, dass folgende Aufgabenbeschreibung existiert:

- Das Ziel der Mission ist es, Fahrzeuge zu detektieren. Nur Fahrzeuge auf einer Straße sind relevant. Die Mission findet um 23:13 Uhr statt.

Die Aufgabe besteht darin, anhand der Aufgabenbeschreibung zu schlussfolgern, welche Elemente aus den definierten Fakten notwendig sind, um das Ziel der Mission zu erfüllen und warum sie notwendig sind.

Die drei Fakten und die Aufgabenbeschreibung können in die Tripel-Darstellungsform *Subjekt-Prädikat-Objekt* umgeschrieben werden. Subjekt und Objekt sind Informationselemente, das Prädikat eine als gültig angenommene Beziehung, die die beiden Elemente miteinander verbindet.

1. Missionsobjekt „ist-ein“ Fahrzeug
2. Missionsmethode „ist“ Detektion
3. Objekt „befindet-sich-auf“ Straße
4. Missionszeitpunkt „ist“ Nacht
5. UAV "ausgestattet-mit“ Sensoren (2)
6. Sensor1 „hat-Typ“ Elektrooptisch
7. Sensor2 „hat-Typ“ Infrarot
8. Sensoren „produzieren“ Sensordaten
9. UAV „ausgestattet-mit“ Computer
10. Computer „führt-aus“ BV-Algorithmus
11. BV-Algorithmus „erstellt“ Fahrzeughypothesen
12. BV-Algorithmus „benötigt“ Sensordaten
13. Fahrzeughypothesen „detektieren“ Fahrzeuge

Jede Aussage repräsentiert für sich einen unabhängigen Fakt. Einige Aussagen sind generell und können auch in anderen Zusammenhängen genutzt werden. Durch die Umstrukturierung folgt nun jede Aussage derselben Syntax. Die Aussagen haben damit eine formale Beschreibung, wie es für die Interpretation durch eine Maschine erforderlich ist.

Ein Mensch ist in der Lage, sowohl aus den in Prosatext formulierten Fakten als auch in der Tripeldarstellung auf Zusammenhänge zwischen Fakten und Aufgabenbeschreibung zu schließen und diese zu interpretieren. Aufgrund seiner Kompetenzen und Erfahrungen kann er, ohne dass dieser Sachverhalt explizit angegeben wurde, schließen, dass z.B. zum erfolgreichen Erfüllen des Ziels aufgrund der Uhrzeit der Infrarotsensor verwendet werden muss. Ein Mensch kann aus dem Beispieltext weitere Zusammenhänge schließen, z B. dass

(1) das Missionsobjekt Fahrzeuge sind,
(2) BV-Algorithmen Fahrzeughypothesen erstellen und durch diese Fahrzeuge detektiert werden.

Auch die Bedeutung der Prädikate ist selbsterklärend.

Für eine Maschine sind die Fakten im Prosatext zunächst nur eine Aneinanderreihung von Zeichenfolgen, ohne dass dabei irgendeine Bedeutung ersichtlich wird. Die Maschine kann keinen Zusammenhang zwischen den Fakten und der Aufgabenbeschreibung entdecken.

Damit ein Computer dieselben Schlüsse ziehen und die Semantik der Aussagen interpretieren kann, muss eine Verbindung zwischen der syntaktischen Darstellung und den daraus folgenden Aktionen hergestellt werden. Durch die Tripel-Darstellungsform (und die Definition von Regeln, auf die in den folgenden Kapiteln näher eingegangen wird) kann die Maschine interpretieren, dass *Fahrzeug* aus Statement 1 dasselbe *Fahrzeug* aus Statement 13 sein muss. *Fahrzeughypothesen* aus Statement 13 muss dieselben *Fahrzeughypothesen* aus Statement 11 sein usw. Aufgrund einer einheitlichen Darstellungsform und einer definierten Semantik ist die Maschine in der Lage, einen Zusammenhang zwischen dem *Missionsobjekt* und der Notwendigkeit von *BV-Algorithmen* zu schließen.

Da die Maschine die Zusammenhänge, die zwischen den Fakten und der Aufgabenbeschreibung bestehen, nun herstellen kann, ist das System in der Lage, Fragen eines Benutzers in Form

von Abfragen zu beantworten. Eine mögliche Fragestellung wäre (unter der Annahme weiterer Fakten) z.B. „Welche Komponenten werden gebraucht, um das Missionsziel zu erfüllen?“. Die Maschine kann nun aufgrund der Statements 1,5, 8,9,10 11,12 und 13 ermitteln, dass die Komponenten

- *UAV*,
- *Sensor* und
- *Computer*

notwendig sind. Auf die Frage „In welchem Zusammenhang stehen *UAV*, *Sensor* und *Computer?*“ wäre die Maschine in der Lage, die folgenden Antworten zu geben:

- UAV ist ausgestattet mit Computer
- UAV ist ausgestattet mit Sensoren
- Computer führt BV-Algorithmen aus
- BV-Algorithmen benötigen Sensordaten
- Sensoren produzieren Sensordaten

Schlussfolgernd kann gesagt werden, dass

> *„… Wissensrepräsentation in deklarativer Weise Aussagen über die zu modellierende Realität darstellen kann. Dazu werden Fakten (in Form von Objekten und Beziehungen) und Regeln formuliert. Hieraus können Inferenzmechanismen weitere Fakten durch Reasoning (Inferenz) ableiten. Die Semantik der Wissensrepräsentation manifestiert sich in den Operationen, die ein Computer auf Basis des repräsentierten Wissens ausführen kann. Die Ausdrucksmächtigkeit einer Wissensrepräsentation ist also untrennbar mit der Leistungsfähigkeit der jeweiligen Inferenzregeln verknüpft. „*[71]

Das folgende Kapitel soll potentielle Formen der Wissensrepräsentation für ein im Kontext dieser Arbeit anwendbares Ressourcenmodell sichten und bewerten (Kapitel 5.2) sowie konzeptuelle Grundlagen für die anschließende Implementierung (Kapitel 5.3.3) liefern.

5.1 Formalismen zur Wissensrepräsentation

Um Wissen formal beschreiben zu können, sodass eine Maschine mit methodischen Vorschriften und formalem Wissen Schlussfolgerungen ziehen kann, sind in den Computerwissenschaften eine Reihe von formalen Repräsentationsformen entwickelt worden.

Ein wichtiges Unterscheidungsmerkmal der verschiedenen Formalismen ist die Art, wie *explizite* und *implizite* Sachverhalte dargestellt werden [69]. Bei der *expliziten* Repräsentation wird das Wissen symbolsprachlich angeordnet. Dieses enthält einzelne Aussagen bzw. Sachverhalte, die als gültig angenommen werden und direkt aus der Repräsentation heraus abgelesen werden können, z.B. „Aus A folgt B“. Ein *impliziter* Sachverhalt muss durch Berechnung oder das Anwenden bestimmter Regeln ermittelt werden, z.B. „Wenn aus A B folgt und aus B C folgt, dann folgt aus A auch C“. Man spricht hier auch vom Schlussfolgern bzw. von Inferenzen.

Welche Art von Inferenz gebildet werden kann, hängt u.a. von den zur Verfügung stehenden Ausdrucksmitteln ab.

Dementsprechend ist die Ausdrucksstärke einer Repräsentationsform ausschlaggebend dafür, ob diese für die Modellierung eines Sachverhaltes in Frage kommt oder nicht. Die bekanntesten davon werden im Folgenden diskutiert. Anschließend werden die Vor- und Nachteile der vorgestellten Formalismen zusammengefasst und bewertet.

Zunächst werden Formalismen aus dem Bereich der mathematischen Logik besprochen. Zu diesen zählen die *Aussagenlogik*, die *Prädikatenlogik* und die *Beschreibungslogik*. Repräsentationsformen mit Fokus auf assoziativen Beziehungen sind die *semantischen Netze* und *Frames*. Ein Formalismus, der auf einer gemeinsamen, fest definierten Konzeptualisierung aufbaut, ist die *Ontologie*. Neben den erwähnten Formalismen gibt es viele weitere, die im Rahmen dieser Arbeit nicht näher betrachtet werden, da sie entweder Ähnlichkeiten (*Regelbasierte Systeme* [69][70], *Topic Maps* [70] [71], *Conceptual Dependencies*[71]) mit den vorgestellten Formalismen haben, teilweise oder gänzlich von den vorgestellten Formalismen abgedeckt werden (*Taxonomien*, *Thesauri* [70] [71]) oder im Rahmen dieser Arbeit nicht relevant erscheinen (*Fuzzy-Logik*[73], *Bayessche Netze*[70]). Eine strikte Trennung der Formalismen ist nicht immer möglich. Teilweise greifen sie ineinander oder können als Fragmente eines anderen Formalismus interpretiert werden.

5.1.1 Aussagenlogik

Logische Formulierungen erlauben es, Schlussfolgerungen aus Wissensbeschreibungen zu ziehen, indem durch Relationen ein Bezug zwischen (symbolischen) Objekten hergestellt wird. In der Aussagenlogik werden Aussagen so formuliert, dass sie als wahr angenommen werden können. Aus wahren Aussagen kann auf eine Lösung für ein Problem geschlossen werden. Logik-Konstrukte sind der Grundstock, um formale Repräsentationssprachen zu entwickeln. Die simpelste, jedoch ausdrucksschwächste, ist die Aussagenlogik.

Die Aussagenlogik (engl. *Propositional Logic*) lässt sich bis in die griechische Antike (Chryssipos von Soli, 281-208 v Chr.) zurückverfolgen. Eine formale Fassung, wie sie heute größtenteils verwendet wird, wurde von Georg Boole 1854 beschrieben. Die Aussagenlogik beschreibt Sätze, die entweder *wahr* oder *falsch* sein können. Die Aussagen können mittels fünf unterschiedlicher Junktoren genauer spezifiziert werden (vgl. Tabelle 1). Dabei benötigt jeder Junktor (bis auf die *Negation*) zwei Aussagen.

Junktor	Name	Bedeutung	Beispiel
$\neg$	Negation	Nicht	$\neg A$
$\wedge$	Konjunktion	Und	$A \wedge B$
$\vee$	Disjunktion	Oder	$A \vee B$
$\rightarrow$	Implikation	Wenn - dann	$A \rightarrow B$
$\leftrightarrow$	Äquivalenz	Genau dann, wenn	$A \leftrightarrow B$

Tabelle 1: Junktoren der Aussagenlogik

Zusätzlich existieren unterschiedliche Erzeugungsregeln (z.B. Klammerregeln) für Sätze und Präzedenzregeln (z.B. Negation vor Konjunktion). Beispiele dafür sind

- „ist x ein Satz, dann ist $\neg x$ ebenfalls ein Satz“,
- „sind x und y Sätze, dann sind auch $x \wedge y, x \vee y, x \rightarrow y, x \leftrightarrow y$ ebenfalls Sätze“,
- etc.

Um Sachverhalte mittels der Aussagenlogik zu formulieren, werden Aussagen, die einen der beiden Wahrheitswerte annehmen können, *Propositionen*[17] zugeordnet. Das nachfolgende Beispiel soll dies verdeutlichen.

- a sei gleichbedeutend mit „Flugzeug hat Startfreigabe“
- b sei gleichbedeutend mit „Pilot ist bereit“
- c sei gleichbedeutend mit „Flugzeug kann starten“

Nun können mittels der Junktoren unterschiedliche Aussagen gemacht werden. Aus den zusammengesetzten Aussagen können durch Inferenzregeln Schlussfolgerungen gezogen werden. Beispiele dafür sind:

- $a \wedge b \rightarrow c$, „Wenn das Flugzeug eine Startfreigabe besitzt und der Pilot bereit ist, dann kann das Flugzeug starten.
- $b \wedge \neg a \rightarrow \neg c$, „Wenn der Pilot bereit ist und das Flugzeug keine Startfreigabe besitzt, dann kann das Flugzeug nicht starten.“

Für einfache Zusammenhänge bzw. Zusammenhänge in einer kleinen abgeschlossenen Domäne ist die Aussagenlogik ausreichend. Möchte man jedoch komplexere Aussagen beschreiben, stößt die Aussagenlogik an ihre Grenzen. Um das Beispiel „*Denis und Christian sind Eurofighter Piloten beim Flugzeughersteller Airbus*“ mittels Aussagenlogik zu beschreiben, müsste man für jeden Fakt eine Aussage formulieren. Dies führt schnell zu einer hohen Anzahl an Aussagemöglichkeiten (da für jede Aussage ebenfalls die 5 Junktoren angewandt werden müssen), für die bewiesen werden muss, dass die Aussagen wahr sind und keine Widersprüche existieren. Zudem können bei einer hohen Anzahl von Aussagen bzw. Propositionen sehr komplexe Terme entstehen, die schwer lesbar sind.

In der Praxis ist es sehr umständlich, Probleme mit der Aussagenlogik zu beschreiben. Die Aussage „Das UAV befindet sich an Koordinate (4E, 12N)“ kann zwar als Variable „UAV_Koordinate_(4E, 12N)“ gespeichert und zum Schlussfolgern verwendet werden. Umständlich wird es jedoch, wenn man alle Positionen (nur im ganzzahligen Bereich), an denen sich das UAV befinden kann, beschreiben möchte. Damit alle Positionen beschrieben werden können, sind 360 * 360 Variablen notwendig. Schwieriger wird es, wenn man z.B. „das UAV befindet sich oberhalb der Basis“ beschreiben will.

[17] Eine Aussage mit einem Wahrheitswert, die meist einer Variable zugeordnet wird.

Eine elegantere Art, Wissen zu repräsentieren, stellt die Prädikatenlogik bereit. Die Ausdrucksmöglichkeiten in der Prädikatenlogik sind die der Aussagenlogik ähnlich. Jedoch besitzt die Prädikatenlogik Sprachkonstrukte, die eine höhere Ausdrucksstärke besitzen.

5.1.2 Prädikatenlogik

Um allgemeinere Sachverhalte ausdrücken bzw. terminologisches Wissen repräsentieren zu können, benötigt man eine Logik mit mehr Ausdruckskraft – die Prädikatenlogik, genauer: die Prädikatenlogik erster Stufe[18] (engl. *First Order Logic*) [74]. Ergänzend zu den Junktoren in der Aussagenlogik kommen in der Prädikatenlogik sog. *Quantoren*, wie sie in Tabelle 2 zusehen, zum Einsatz. Quantoren werden genutzt, um Aussagen über eine Menge von Objekten zu machen. Dabei müssen diese nicht explizit auf gezählt werden.

Quantor	Name	Bedeutung
$\exists$	Existenzquantor	Es existiert
$\forall$	Allquantor	Für alle

Tabelle 2: Quantoren in der Prädikatenlogik

Zusätzlich gibt es in der Prädikatenlogik Variablen (meist durch Großbuchstaben abgebildet, z.B.X, Y, Z), Symbole für Konstanten (durch Kleinbuchstaben repräsentiert, z.B.a, b, c), Funktionssymbole f, g, h und Relations- bzw. Prädikatssymbole p, q, r. Mit den Quantoren lassen sich Mengen zusammenfassen und allgemeine von spezifischen Aussagen trennen. Eine allgemeine Aussage in der Prädikatenlogik kann folgendermaßen formuliert werden:

- $\exists X: Eurofighter(X)$, „X ist ein Eurofighter."
- $\forall X: Flugzeug(X) \rightarrow fliegt(X)$, „Alle Flugzeuge fliegen."
- $\forall X\ \forall Y: Pilot(X,Y) \leftrightarrow (\ navigiert(X,Y)\ \wedge Operateuer(X)$, „Der Pilot eines UAVs ist der navigierende Operateur."
- $\exists X: UAV(X) \wedge solarbetrieben(X)$, „Es gibt UAVs, die solarbetrieben sind."

Die Aussagen sind nicht auf zwei Prädikate beschränkt, sondern lassen sich zu einem n-stelligen Prädikat ausbauen. Außerdem lassen sich mit der Prädikatenlogik Eigenschaften wie die *Transitivität* und *Symmetrie* ausdrücken [69], [75]:

- Transitivität: Sei X ein Eurofighter, Y der Flugzeugtyp Jet und Z die Klasse Luftfahrzeug so gilt: $\forall X, Y, Z : is_a\ (X,Y) \wedge is_a(Y,Z) \rightarrow is_a(X,Z)$, „Wenn Eurofighter ein Jet ist und Jet ein Luftfahrzeug, dann ist Eurofighter ebenfalls ein Luftfahrzeug."
- Symmetrie: $\forall\ X, Y : bruder(X,Y) \rightarrow bruder(Y,X)$, „Der Bruder eines Bruders ist wiederum dessen Bruder."

[18] Logik höherer Ordnung ist aufgrund der hohen Berechnungskomplexität und der Unentscheidbarkeit im Rahmen dieser Arbeit nicht von Interesse.

Syntaktisch gültige Formeln für die Prädikatenlogik können folgendermaßen zusammengefasst werden [71]:

- Eine Konstante oder eine Variable ist ein Term.
- Eine auf n Terme angewandte n-stellige Funktion ist wieder ein Term.
- Ein auf n Terme angewandtes n-stelliges Prädikat entspricht einer korrekten Formel.
- Eine logische Kombination von korrekten Formeln entspricht wiederum einer korrekten Formel, wobei alle logischen Junktoren der Aussagenlogik erlaubt sind
- Eine korrekte Formel kann in eine neue Formel überführt werden, indem man einen Quantor und eine Variable voranstellt.

Die Aussagenstärke ist hinreichend, um Fakten formal zu beschreiben und darauf Inferenzregeln anzuwenden. Kritisch ist jedoch die sogenannte *Resolution*, das Verfahren, welches das automatische Schlussfolgern ermöglicht und Aussagen auf Konsistenz prüft. Das Resolutionsverfahren erlaubt es, Wiedersprüche aus verschiedenen Aussagen abzuleiten, indem Konsequenzen entweder als *wahr* oder *falsch* interpretiert werden. Führt man mindestens einen weiteren Wahrheitswert, z.B. „inkonsistent", ein, so spricht man von einer mehrwertigen Logik. Eine mehrwertige Logik kann z.B. eingesetzt werden, um widersprüchliches, unsicheres und ungenaues Wissen zu beschreiben. Eine Erweiterung der mehrwertigen Logik (um mit ungenauem Wissen umgehen zu können) ist die *Fuzzy-Logik*. Man schreibt $T \vDash F$ für eine Konsequenz, die wahr ist und $T \nvDash F$ für eine Konsequenz, die nicht wahr, bzw. falsch ist. Eine (logische) Konsequenz in der Prädikatenlogik besteht aus einer Theorie T (gleichbedeutend mit einer Menge von Formeln) und einer Interpretation I, die ein Modell von T ist, wenn $I \vDash G$ für jede Formel G in T gilt. Das bedeutet, dass für eine Formel F gilt, dass diese eine logische Konsequenz aus T ist, für den Fall, dass jedes Modell von T auch ein Modell von F ist, sodass gilt: $T \vDash F$. Das Problem in der Prädikatenlogik ist, dass die Resolution nicht entscheidbar ist, d.h. gibt es innerhalb einer Menge keine Widersprüche, lässt sich die Terminierung der Resolution nicht garantieren, es entsteht eine unendliche Schleife. Die Prädikatenlogik ist damit unentscheidbar, kann jedoch semi-entscheidbar werden, wenn diese beschränkt (vgl. Beschreibungslogiken in Kapitel 5.1.5) wird. Das heißt, es kann immer in einer endlichen Zeit nachgewiesen werden, dass $T \vDash F$ gilt, nicht aber $T \nvDash F$. Eine logische Konsequenz kann also immer in endlicher Zeit nachgewiesen werden, der umgekehrte Fall trifft jedoch nicht zu. Da die Entscheidbarkeit für ein praktisches System unabdingbar ist und die Prädikatenlogik über die nötige Aussagekraft verfügt, wird versucht, Fragmente aus der Prädikatenlogik zu extrahieren, die die nötige Aussagestärke besitzen, jedoch immer entscheidbar sind. Zu diesen zählen u.a. die *Beschreibungslogiken*, auf die in Kapitel 5.1.5 eingegangen wird. Andere entscheidbare Teilmengen der Prädikatenlogik sind z.B. die zuvor vorgestellte *Aussagenlogik*, *Disjunktives Datalog* und die *Hornklauseln* [41].

Was jedoch fehlt, ist die Möglichkeit, Wissen zu organisieren. Zur Beschreibung von Wissen in einem kontextuellen bzw. sachlichen Zusammenhang eignen sich *Semantische Netze*.

5.1.3 Semantische Netze

Semantische Netze (begründet von [76]) bedienen sich einer graphischen Notation, um die Bedeutung von Begriffen durch assoziative Beziehungen zu anderen Objekten zu beschreiben. Dabei müssen die Begriffe nicht zwingend in einem Verhältnis stehen, können aber in bestimmten Fällen in eine lose Beziehung gebracht werden.

Sie wurden ursprünglich für kognitive Strukturen, wie z.B. der Darstellung von Assoziationen im Gedächtnis und der formalen Darstellung von Sätzen natürlicher Sprache, genutzt [71]. Ein einfaches semantisches Netz, bestehend aus einem Subjekt-Prädikat-Objekt Tripel, für den Satz „Denis S. ist Pilot“ ist in Abbildung 28 zusehen.

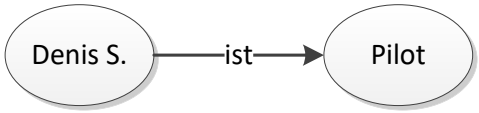


Abbildung 28: Semantisches Netz für den Satz "Denis S. ist Pilot"

Zusammenhangswissen wird als Netzkonstrukt, gebildet aus Objekten und ihren Relationen, aufgespannt. Objekte (z.B. Individuen oder Konzepte) werden als Knoten dargestellt, die Verbindungen bzw. Beziehungen zwischen diesen als Kanten. Die Verbindung von Objekten kann sowohl bidirektional als auch gerichtet sein. Man spricht bei zweistelligen Relationen auch von *assoziativen Beziehungen*.

Mit semantischen Netzen ist es nicht nur möglich, semantische Beziehungen zwischen zwei Objekten zu modellieren (Flugzeug – „hat-teil“ – Propeller), sondern ebenfalls Eigenschaften über Konzepte (Propeller – „Funktion“ – sich drehen) abzubilden. Weiterhin bieten semantische Netze die Möglichkeit, Konzeptklassen und Individualkonzepte abzubilden. Knoten können dabei entweder eine Klasse (Flugzeug) oder ein Individuum („Eurofighter E1“) beschreiben. So ist es möglich, Aussagen über z.B. Eigenschaften für eine ganze Klasse oder für ein einzelnes Individuum zu treffen. Abbildung 29 zeigt ein einfaches semantisches Netz für den Begriff „Flugzeug“.

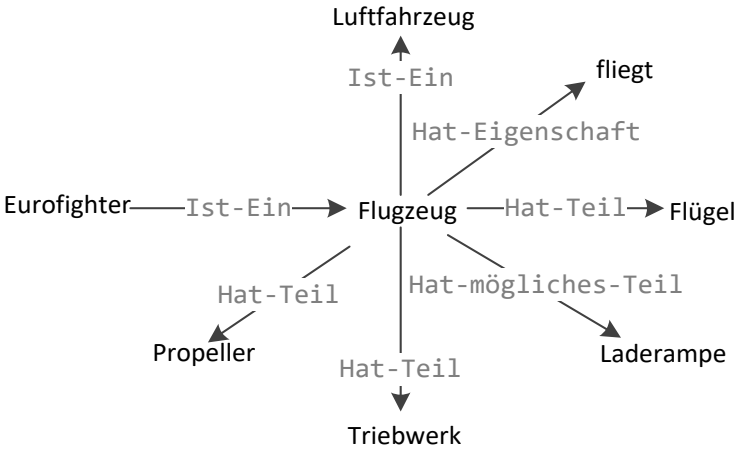


Abbildung 29: Beispiel eines einfachen semantischen Netzes

Wesentliche Eigenschaften von semantischen Netzen sind *Reflexivität*, *Symmetrie* und *Transitivität* [69], [71]:

- Eine zweistellige Relation R heißt *reflexiv* für eine Menge von Objekten, wenn jedes Element der Menge in Relation zu sich selbst steht, also wenn $x\ R\ x$ für alle Elemente x der Menge gilt.
- Eine zweistellige Relation R heißt *symmetrisch* für eine Menge von Objekten, wenn aus $x\ R\ y$ stets $y\ R\ x$ folgt.
- Eine zweistellige Relation R heißt *transitiv* für eine Menge von Objekten, wenn aus $x\ R\ y$ und $y\ R\ z$ stets $x\ R\ z$ folgt.

Eine der wichtigsten Relationen ist die Vererbungsrelation zur Beschreibung von Unter- und Oberbegriffen (*Hyponyme* und *Hyperonyme*). Relationen, auch *epistemische Primitiven* genannt, die allgemeingültig sind und domänenunabhängig außerhalb eines begrenzten Weltausschnitts benutzt werden, sind die

- „ist-Teil"-Beziehung für *transitive* und *asymmetrische* Beziehungen,
- „ist-Teil-von"-Beziehung zur Beschreibung von semantischen Verhältnissen von zwei selbständigen Objekten. Man spricht auch von *Aggregation* bzw. *Partonomie* oder *Meronymie*,
- „hat-Teil"-Beziehung als Gegenpart zu der *Aggregation*. Man spricht auch von *Holonymie*.

Es lassen sich zwei Arten von Netzwerken unterscheiden, *assoziative* und *semantische Netzwerke*[19] [69]. Bei den ersteren gibt es keinen Unterschied zwischen den assoziativen Beziehungen, sprich jeder Kantentyp ist identisch. Bei der anderen Art sind die Kanten beschriftet, das heißt, es gibt verschiedene Typen von Beziehungen (Kanten), je nachdem welcher Zusammenhang modelliert werden soll (siehe Abbildung 29). Während bei assoziativen Netzwerken die Kanten ungerichtet sind, da sie als symmetrische Beziehungen zu interpretieren sind, haben semantische Netze gerichtete Kanten, was mit nicht-symmetrischen Beziehungen gleichzusetzen ist.

Im Gegensatz zur logischen Darstellungsform wirken semantische Netze mit ihrer graphischen Darstellung intuitiver. Außerdem ist es einfach, ein Konzept zu erweitern, sofern es für das Konzept schon einen Knoten gibt. Weitere Aussagen werden schlicht über neue Relationen dem Netz hinzugefügt. Über die Vererbung können Objekte auf einer hohen Ebene beschrieben und die Eigenschaften anschließend an andere Objekte weitergegeben werden. Es findet eine klare Trennung zwischen Objekten, Konzepten und Individuen (in Form von Knoten) und deren Beziehung (in Form von Kanten) statt. Ein weiterer Vorteil der graphischen Notation ist die Fähigkeit, semantische Nähe auszudrücken. Je länger eine Kantenfolge wird, desto geringer ist die semantische Nähe zwischen zwei Objekten.

[19] Eine ganz eindeutige Terminologie existiert nicht, in der Literatur werden die Begriffe unterschiedlich angewandt.

Während in der Prädikatenlogik (höherer Ordnung) das Darstellen mehrstelliger Prädikate kein Problem ist, müssen diese bei den semantischen Netzen auf zweistellige Relationen zurückgeführt werden. Die Aussage „Der Pilot übergibt das Steuer an den Co-Piloten“ würde man mit semantischen Netzen gemäß Abbildung 30 ausdrücken.

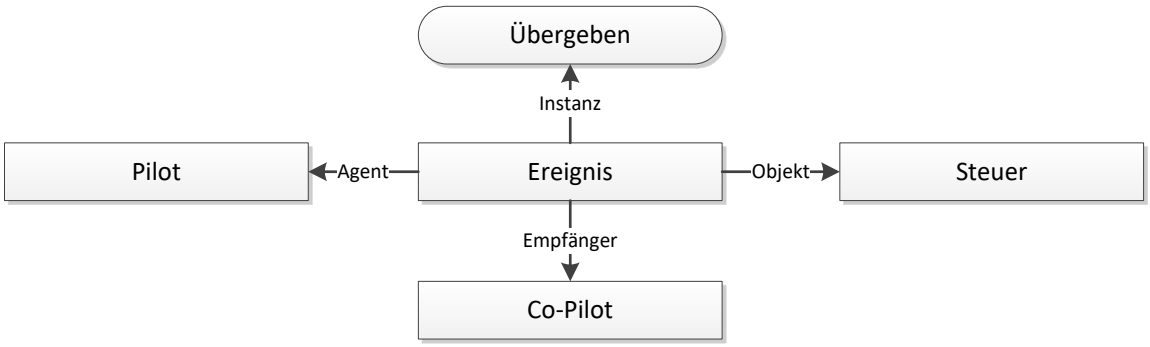


Abbildung 30: Dreistellige Aussage in semantischen Netzten (nach [77]**)**

Nachteile des Vererbungsprinzips werden dann deutlich, wenn Ausprägungen bzw. Eigenschaften eines Objekts von der Norm abweichen. In Abbildung 29 wurde (fälschlicherweise) definiert, dass jedes *Flugzeug* einen *Propeller*[20] hat und dass *Eurofighter* ein *Flugzeug* ist. Dies würde bedeuten, dass jeder Eurofighter einen Propeller hat. Natürlich kann man das Konzept *Flugzeug* noch einmal in Subkonzepte spezialisieren, von denen ein Subkonzept als Teil einen Propeller besitzt und das andere nicht. Solche Maßnahmen führen jedoch zu einer Verkomplizierung des gesamten semantischen Netzes. Weitere Nachteile sind:

- das Fehlen von Negation, Disjunktion und Quantifizierung,
- schnelles Wachstum bei bereits niedriger Komplexität,
- die Herleitung von Wissen kann eine komplexe Suche nach sich ziehen [71],
- fehlende Mechanismen zur Konsistenzprüfung,
- fehlende Semantik. Knoten z.B. können sowohl das Objekt selbst sein oder eine Eigenschaft des Objekts.

5.1.4 Frames

Frames sind ebenso wie Semantische Netze angelehnt an das kognitionspsychologische Modell des menschlichen Gedächtnisses. Während den Semantischen Netzen ein assoziatives Modell zugrunde liegt, nutzten Frames schemaartige Modelle. Den Begriff Frame führt 1974 Marvin Minsky in seiner Arbeit „A framework for representing knowledge“ unter dem Leitgedanken Aristoteles „Was ist da und was kann ich über das sagen, was da ist“ ein. Die Knoten der Frames stellen die Schemata dar, die Kanten die Wechselwirkung zwischen diesen.

Der Begriff Schema wurde von Bartlett 1932 [71] als Bezeichnung für „*... das Wissen von Menschen, etwa über ein Individuum, ein Objekt, ein Ereignis oder eine Handlung*“ eingeführt.

[20] Dieser Umstand ist natürlich nicht korrekt soll jedoch der in diesem Beispiel als wahr angenommen werden.

Schemata sind Modelle stereotypischer Erinnerungsmuster (im Verglich zu reinen Assoziationen zwischen Begriffen). D.h. sie stellen Denkprozesse durch die Repräsentation des Zusammenspiels komplexer Netze mentaler Konzepte nach. Man unterscheidet sechs Merkmal für Schemata (nach Mandel und Spada 1988 [71]):

- Schemata sind kognitive Strukturen, in denen allgemeines Wissen im Gedächtnis repräsentiert ist. Das Wissen über typische Zusammenhänge in einem Realitätsbereich ist in Schemata organisiert.
- Schemata weisen Variablen auf, die unterschiedliche Werte annehmen können
- Schemata können ineinander eingebettet sein
- Schemata enthalten sowohl episodisches als auch generisches Wissen
- Schemata haben sowohl eine Struktur- als auch eine Prozesskomponente
- Schemata repräsentieren Wissen unterschiedlicher Inhaltsbereiche

Frames sind Konzeptrepräsentationen, die die Eigenschaften und die Bedeutung eines Objekts beschreiben. Die Eigenschaften können einen Wert annehmen und werden durch Prädikate oder Attribute beschrieben. Neben einer rein systematischen Objektbeschreibung ist man mit Frames in der Lage, zeitliche Charakteristika und Prozesse abzubilden. *„Frames sind also nichts anderes als modulare Beschreibungen deklarativer und prozeduraler Aspekte unserer Wahrnehmung, die sich im kognitiven Zusammenhang abrufen lassen*“[71]. Frames werden als abstrakte Informationseinheiten gesehen, mit deren Hilfe sich Wissen strukturell beschreiben lässt. Durch Vernetzung der Frames können Zusammenhänge und Wechselwirkungen repräsentiert werden.

Der Sachverhalt in Abbildung 31, abgebildet durch ein semantisches Netz, ist in Tabelle 3 in der Frame-Darstellung dargestellt.

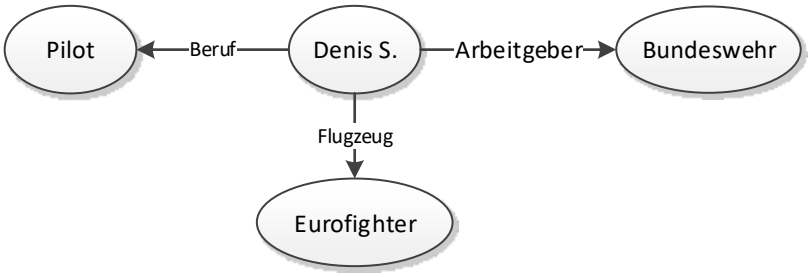


Abbildung 31: Ein einfaches semantisches Netz

Denis S.	Beruf		Arbeitgeber		Flugzeug	
	Pilot	…	**Bundeswehr**	…	**Eurofighter**	…

Tabelle 3: Beispiel für eine Frame-Darstellung

Pilot, Bundeswehr und Eurofighter sind dabei wiederum eigene Frames, die wiederum andere Beschreibungen enthalten können. Die Eigenschaften des Frames *Eurofighter* können z.B. durch Tabelle 4 repräsentiert werden.

Eurofighter	Kosten	Typ
	90 Millionen $	Kampfflugzeug

Tabelle 4: Frameartige Beschreibung des Eurofighter

Frames unterstützen das Abbilden von Klassen, indem „wertlose“ Frames geschaffen werden, welche für eine Menge von Objekten Eigenschaften definieren. Man spricht auch von *Klassifikation*. Klassen können instanziiert werden, um ein ganz konkretes Objekt aus der Realität abzubilden (z.B. den Eurofighter mit der Identifikationsnummer 462-E). Weiterhin ist es möglich zu generalisieren, also das Verknüpfen eines Objektes mit dessen *Hyperonym* bzw. *Hyponym.* Ebenso wie semantische Netze, unterstützen Frames ebenfalls Vererbung, wodurch Beschreibungen von einem Frame auf einen anderen übertragen werden können.

Eigenschaften von Objekten werden bei Frames in sogenannten Slot-Einträgen (vgl. Tabelle 4) dargestellt. Man unterscheidet zwischen *terminalen* und *nicht-terminalen* Slots. Diese können sowohl *einwertig* als auch *mehrwertig* sein und werden nochmals in *Rollen-Slots* und *May-be-a-Slot(s)* unterschieden. Detailliertere Informationen über die Unterschiede der Slots finden sich in [69]. Slots nutzen Kardinalitäten, Assoziationen und Relationen.

Frames unterscheiden sich von logikbasierten Ansätzen und semantischen Netzen durch eine starke Vorstrukturierung von Wissen, da alle Angaben über die Eigenschaften eines Objektes und dessen semantische Beziehungen zusammengefasst sind, sodass bei der Auswahl eines Frames in der Wissensdatenbank keine weiteren Suchprozesse stattfinden müssen. Die Objektzentrierung bei Frames ist höher als bei semantischen Netzen (während die Objektzentrierung bei semantischen Netzen höher ist als bei logikbasierten Ansätzen).

Ein gravierender Nachteil vieler frameartiger Systeme ist der Mangel an bzw. das Fehlen formaler Semantik, was wiederum zu inkonsistenten Wissensdatenbanken führt. Des Weiteren mangelt es Frames an Repräsentationsmöglichkeiten, um allgemeine regelhafte Zusammenhänge und Einschränkungen zu modellieren. Bei den semantischen Netzen wurde bereits erwähnt, dass jede prädikatenlogische Beschreibung erster Ordnung auch durch ein semantisches Netz abgebildet werden kann, indem man Meta-Sprach Relationen einführt. Dies gilt ebenfalls für Frames, hat jedoch denselben Nachteil wie bei den semantischen Netzen. Um den genannten Nachteilen entgegenzuwirken, werden Frames inzwischen oft mit einem zweiten Repräsentationsformat (meistens Logik) verknüpft. Man spricht von *hybriden Frame-Repräsentationen*. Dadurch versucht man die Vorteile z.B. der Prädikatenlogik mit denen der Frames zu verknüpfen [69].

5.1.5 Beschreibungslogiken

Die bisher vorgestellten Formalismen zur Wissensrepräsentation sind mit der Beschreibung von Objekten und deren Eigenschaften offensichtlich geeignet, um menschliches Denken und Wissen zu repräsentieren. Aus computerorientierter Sicht haben sie jedoch den Nachteil, dass das Repräsentationsmodell von der Interpretationsleistung der Datenstrukturen des arbeitenden Programms abhängig ist. Für ein effizientes Arbeiten mit wissensbasierten Systemen bedarf es

einer wohldefinierten Semantik, die sowohl von einem Menschen als auch von einer Maschine interpretiert werden kann. Um dies zu erreichen, ist ein Formalismus erforderlich, der über genügend Ausdrucksstärke besitzt, jedoch nicht zu ausdrucksstark ist, sodass die Berechnungen der impliziten Beziehungen von den Inferenzmechanismen durchführbar sind. Frames und Semantischen Netzen fehlt die notwendige Semantik, sodass man nicht feststellen kann, ob eine Schlussfolgerung korrekt, vollständig oder entscheidbar ist. Um dies zu gewährleisten, sind die *Beschreibungslogiken* [78] bzw. *terminologischen Logiken* (engl. *Description Logics*), von denen die meisten Untermengen der Prädikatenlogik sind, entstanden.

Beschreibungslogiken repräsentieren implizites und explizites Wissen einer Domäne, indem sie mit Hilfe von Konstruktoren bzw. Operatoren Konzepte und Rollen ausdrücken. Sie gehören der Familie der logikbasierten Formalismen an und sind meist auf ein- und zweistellige Prädikate beschränkt. Die Expressivität der Operatoren wird derart beschränkt, dass die Erfüllbarkeit bis auf wenige Ausnahmen gegeben ist.

- Konzepte (unäre Prädikate) beschreiben Klassen, z.B. „Flugzeug“, „Pilot“, etc. ($Flugzeug: \{x \mid Flugzeug(x)\}$).
- Rollen (binäre Prädikate) beschreiben Eigenschaften und Beziehungen, z.B. „fliegt“, „hat Starterlaubnis“, etc. ($fliegt: \{(x, y) \mid fliegt(x, y)\}$).

Weiterhin können mittels Beschreibungslogiken Individuen modelliert werden, also konkrete Instanzen von Klassen, z.B. Pilot (Denis S.). Aus einfachen Beschreibungen von Konstruktoren werden komplexere Beschreibungen generiert. Dabei gibt es eine Vielzahl von Konstruktoren, die sich in ihrer Ausdrucksstärke unterscheiden. Je nachdem welche Konstruktoren von einer Beschreibungslogik verwendet werden, kann die Komplexität der zu beschreibenden Sachverhalte steigen oder sinken. Neben den Konstruktoren gibt es eine Menge von Axiomen, die verwendet werden, um Fakten über Konzepte, Rollen und Instanzen auszudrücken. Beschreibungslogiken haben eine größere Ausdrucksstärke als die *Aussagenlogik*, sind jedoch ausdrucksschwächer als die *Prädikatenlogik*. Entstanden ist die Beschreibungslogik aus den *semantischen Netzwerken* und weist Ähnlichkeiten zur *Modallogik* auf.

Die allgemeine Architektur von Beschreibungslogiken besteht aus einer TBox und einer ABox [79], [80] (vgl. Abbildung 32).

Die *TBox* enthält das terminologische Wissen, d.h. Wissen über die Konzepte einer Domäne. Zu diesen gehören unter anderem die Klassen, Attribute und Eigenschaften, aber auch die Beziehungen und Einschränkungen der Klassen. Die *ABox* enthält das assertionale Wissen, also Wissen über die Instanzen einer Ontologie. Die *TBox* und *ABox* befinden sich innerhalb der Wissensbasis, an die eine Inferenzmaschine angeschlossen ist. Die Inferenzmaschine wird genutzt, um Schlussfolgerungen aus den Informationen in der *TBox* und der *ABox* zu ziehen. Zuletzt gibt es ein Interface, um den Zugriff auf die Informationen zu ermöglichen.

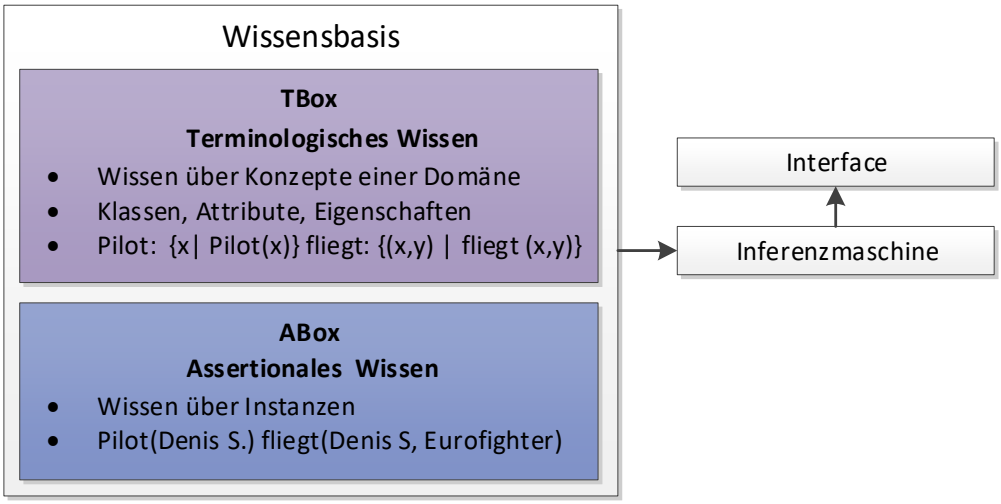


Abbildung 32: TBox und ABox

Jede Beschreibungslogik beinhaltet die bereits kennengelernten Operatoren:

- Konjunktion ($\sqcap$),
- Disjunktion ($\sqcup$) und
- Negation ($\neg$)

als Klassenkonstrukt. Quantoren, wie sie bei der *Prädikatenlogik* vorgestellt wurden, werden verwendet, um ausschließlich Rollen zu beschreiben und Rollenbereiche (*Domain*, *Range*) einzuschränken.

$$Pilot \sqcap \exists fliegtFlugzeug.Flächenflugzeug$$

Das Beispiel verdeutlicht den Einsatz der Quantoren. Die oben dargestellte Aussage bedeutet, dass es einen Piloten gibt, der ein Flugzeug fliegt. Das Flugzeug hat jedoch die Einschränkung, dass es ein Flächenflugzeug sein muss. Sinngemäß würde das heißen, dass der Pilot z.B. nicht in der Lage ist, einen Hubschrauber zu fliegen.

Weitere Konstruktoren (unabhängig einer konkreten Beschreibungslogik) sollen im Folgenden kurz zusammengefasst werden. Nicht alle Beschreibungslogiken müssen alle genannten Konstruktoren unterstützen.

- Zahlenrestriktionen: ≤ 1 hatSitze, $\leq n\ P = \{x\ |P(x,y)| \leq n\}$ gesucht sind alle Luftfahrzeuge, die einen oder keine Sitze haben.
- Qualifizierte Zahlenrestriktionen: Eine Zahlenrestriktion, bei der Rolleneinschränkungen gelten, z.B. hatSitze.Flächenflugzeug, was gleichbedeutend mit der Aussage ist, dass die Zahlenrestriktion nur für die Luftfahrzeuge gilt, die vom Typ Flächenflugzeug sind.
- Nominale: Zur Definition von Klassen, die eine bestimmte Anzahl von Instanzen haben, z.B. $\{Tornado, Eurofigther, Su-27\} = Jet - Flugzeuge$

- Restriktionen (für Datentypen): Restriktionen schränken den Wertebereich einer Rolle ein, z.B. könnte die Rolle „maxFlughöhe“ auf den minimalen Wertebereich 0 eingeschränkt werden, sodass es nicht möglich ist, die Flughöhe auf einen negativen Wert zu setzen.
- Inverse Rollen: $fliegtFlugzeug^{-} \equiv hatPilot$
- Transitivität: $bestehtAus \sqsubseteq +bestehtAus$, ein Teil besteht aus einem Teil, welches wiederum aus einem Teil besteht.

Die kleinste Beschreibungslogik, die aussagenlogisch abgeschlossen ist, wird $\mathcal{ALC}$ (*Attribute Language with Complement*) genannt. Eine Übersicht über alle Logiken und deren Konstruktoren ist in Tabelle 5 [101] zusehen.

Operator / Konstruktor	Syntax	Sprache	
Konjunktion	$A \sqcap B$	$\mathcal{FL}$	$S*$
Wertrestriktion	$\forall R.C$		
Existenzquantor	$\exists R$		
Top (allgemeines Konzept)	$\top$	$\mathcal{AL}*$	
Bottom (spezielles Konzept)	$\bot$		
Negation ($\mathcal{C}$)	$\neg A$		
Disjunktion	$C \sqcup D$		
Existentielle Restriktion	$\exists R.C$		
Zahlenrestriktion ($\mathcal{N}$)	$(\leq n\ R)(\geq n\ R)$		
Mengen von Individuen ($\mathcal{O}$)	$\{a_1, \dots a_n\}$		
Beziehungshierarchie	$R \subseteq S$	$\mathcal{H}$	
Inverse Beziehungen	R^{-1}	$\mathcal{I}$	
Qualifizierte Zahlenrestriktion	$(\leq n\ R.C)(\geq n\ R.C)$	$\mathcal{Q}$	

Tabelle 5: Beschreibungslogiken und Ihre Charakteristika[21]

Die Bedeutung der Symbole in der Spalte „Sprache“ sollen im Folgenden kurz zusammengefasst werden. Auf diese wird im Kapitel 5.3.2.2 noch einmal verwiesen.

- $\mathcal{ALC}$: *Attribute Language with Complement*
- S: $\mathcal{ALC}$ + Rollentransitivität
- $\mathcal{H}$: Subrollenbeziehung
- $\mathcal{O}$: *A*bgeschlossene Klassen
- $\mathcal{I}$: Inverse Rollen
- $\mathcal{N}$: Zahlenrestriktionen
- $\mathcal{Q}$: Qualifizierende Zahlenrestriktionen
- $\mathcal{F}$: Funktionale Rollen
- $(\mathcal{D})$: Datentypen

[21] Abbildung entnommen aus der Vorlesung „Semantic Web Technologien“, Dr. H. Sack, Hasso-Plattner-Institut für Softwaresystemtechnik, Universität Potsdam, Wintersemester 2011/12, veröffentlicht am 29.11.2011

Die einfachste Beschreibung ist $\mathcal{FL}$ (*Frame Logic*), die als Elemente die Konjunktion, die Wertrestriktion und den Existenzquantor besitzt. Ausdrucksstärkere Beschreibungen sind in den $\mathcal{AL}$ (Attribute Language) Sprachen (vgl. mit $\mathcal{ALC}$- Das $\mathcal{C}$ resultiert aus der Negation, Tabelle 5) enthalten. Zusätzlich zu $\mathcal{FL}$ kommen hier noch die *Top*, *Bottom*, *Negation*, *Disjunktion* und die *existentielle Restriktion* hinzu. Interessant für diese Arbeit ist die Beschreibungslogik $\mathcal{SHOIN}$, auf die später noch genauer eingegangen wird.

Beschreibungslogiken modellieren Objekte und Domänen, indem komplexe Beziehungen und Ausdrücke durch atomare Konzepte und Rollen der zu beschreibenden Elemente ausgedrückt werden. Sie besitzen (im Gegensatz zu Frames und Semantischen Netzen) eine wohldefinierte Semantik, sodass eine strukturelle Unterscheidung zwischen Terminologie und Domäne stattfindet.

5.1.6 Ontologien

Mit Beschreibungslogiken findet zum einen eine strukturelle Unterscheidung zwischen Terminologie und Domäne statt, zum anderen können komplexe Zusammenhänge semantisch beschrieben werden. Um eine erfolgreiche Kommunikation, nicht nur zwischen Menschen, sondern besonders zwischen Computersystemen, zu gewährleisten, sind feststehende Begriffe, die von beiden Kommunikationsseiten identisch interpretiert werden, notwendig. Jeder Begriff besitzt umfangreiches, zum Teil spezifisches, Wissen über eine bestimmte Gegebenheit, Abhängigkeiten, oder die aktuelle Situation, was für ein gemeinsames Verständnis notwendig ist.

Die Idee, Wissen über Begriffe und die damit verbundenen Konzepte formal und präzise darzustellen, sodass Computer mit den Informationen arbeiten und diese interpretieren können, hat ihren Ursprung in der Philosophie der Antike: die *Ontologie*.

Der Begriff *Ontologie* stammt aus dem griechischen (altgriechisch ὄν ón ‚seiend', und -logie (aus λόγος lógos „Lehre“)) und beschreibt in der Philosophie die Lehre vom Seienden. Mit anderen Worten ausgedrückt, beschreibt der Begriff das Erklären und Beschreiben der Welt an sich, so wie sie ist und nicht so, wie man sie als Mensch mittels der Sinneswahrnehmung und des Verstandes wahrnimmt. Andere Bereiche, in denen das Paradigma der Ontologien genutzt wird, sind Linguistik, Informationsverarbeitung und Informatik. Der Begriff wurde von zahlreichen Philosophen (z.B. Sokrates, Aristoteles, Thomas von Aquin, Jacob Lorhard, Christian Wolff, Immanuel Kant) aufgegriffen. Im Rahmen dieser Arbeit soll die Sichtweise der Informatik auf den Ontologiebegriff näher erörtert werden.

Ursprünglich verwendet ausschließlich in Anwendungsgebieten der Künstlichen Intelligenz, wurden Ontologien später im Rahmen des *Semantic Webs* im Bereich des World Wide Web eingesetzt, später im Bereich der Medizin[22] [82]. Heute findet man immer mehr Anwendungsgebiete, in denen der Einsatz von Ontologien sich als vorteilhaft herausstellt [83]. Über die Jahre haben sich dabei die Sprachen RDF, RDFS und besonders OWL als Ontologiesprachen

[22] https://bioportal.bioontology.org/ontologies/SNOMEDCT, http://www.imise.uni-leipzig.de/Gruppen/OntoMed/index.jsp

(siehe Kapitel 5.3.2), welche allesamt auf dem XML Grundgerüst basieren, durchgesetzt. Gerade bei der Entwicklung im *Semantic Web* hat sich RDFS bzw. OWL bewährt, wodurch gerade OWL zur meistverwendeten Ontologiesprache wurde.

In der Regel werden Ontologien genutzt, um (Teil-)Wissen domänenspezifisch zu beschreiben. Ziel ist es, eine gemeinsame Konzeptualisierung zu schaffen, die als Grundlage für Kommunikation genutzt werden kann. Das heißt, Ontologien werden immer dann verwendet, wenn Informationen bzw. Daten von mehreren Kommunikationspartnern mit unterschiedlichem Wissen verwendet werden sollen. Dabei wird versucht, Wissen über Begriffe und deren Zusammenhänge (die Konzepte) in einer formalen Weise zu beschreiben, sodass Computer dieses Wissen nutzen können. D.h., dass jede der in Kapitel 5.1 vorgestellten Formalismen zur Modellierung einer Ontologie genutzt werden kann (abhängig von der benötigten Ausdrucksstärke bzw. Komplexität des zu modellierenden Wissens). Der am häufigsten genutzte Formalismus in Kombination mit Ontologien ist jedoch die Beschreibungslogik. Für verschiedene Domänen können eine oder mehrere Ontologien existieren, die verschiedene Konzeptualisierungen beschreiben. Das Domänenwissen wird in Form eines von allen Teilnehmern akzeptierten Begriffslexikons repräsentiert, sodass dessen Wiederverwendung von verschiedenen Anwendergruppen ermöglicht wird.

5.2 Formalismenbewertung- und Auswahl

In Kapitel 5.1 wurden verschiedene Formalismen zur Repräsentation von Wissen vorgestellt. Nicht jede der vorgestellten Formalismen ist jedoch geeignet, das im Rahmen dieser Arbeit diskutierte Ressourcenmodell samt der in Kapitel 1.4 aufgestellten Anforderungen zu beschreiben.

Die Formalismen sollen im Folgenden miteinander verglichen und bewertet werden. Hierzu wurden verschiedene Eigenschaften gewählt, die eine Differenzierung der Formalismen im Rahmen dieser Arbeit erlauben. Es wurden bewusst nicht alle Eigenschaften der unterschiedlichen Formalismen aufgezählt (z.B. Transitivität, etc.), da dies an dieser Stelle nicht zielführend ist.

Die Bewertung findet anhand folgender Eigenschaften statt:

- **Klassen, Instanzen, Beziehungen**: Es muss die Möglichkeit geben, Klassen zu erstellen, diese mit Beziehungen in Relation zu setzen und konkrete Instanziierungen davon zu erstellen.
- **Mengen, Relationen**: Es muss die Möglichkeit bestehen, Mengen zu beschreiben (z.B. Fähigkeiten und Ressourcen).
- **Entscheidbar**: Das Ressourcenmodell muss entscheidbar sein, damit die Softwareverarbeitung in einer endlichen Zeit abgeschlossen ist (vgl. Kapitel 1.4.1).
- **Schlussfolgerungen**: Um implizites Wissen und dynamische Zusammenhänge z.B. zwischen Fähigkeiten und Ressourcen modellieren zu können, sind Inferenzmechanismen notwendig.

- **Formale Semantik**: Um Ressourcen und Fähigkeiten in einer festen Domäne beschrieben zu können, ist eine formale Semantik notwendig.
- **Regeln**: zur Beschreibung von Vor- und Nachbedingungen von Ressourcen soll es die Möglichkeit geben, Regeln benutzen zu können.
- **Restriktion**: Um Limitationen von Ressourcen beschreiben zu können, werden Restriktionen benötigt.
- **Konsistenz**: Die Möglichkeit von Konsistenzprüfungen soll garantieren, dass Zusammenhänge logisch korrekt modelliert wurden.
- **Konzeptualisierung**: Schaffen eines gemeinsamen Verständnisses von Begriffen zum Austausch von Informationen zwischen verschiedenen Kommunikationspartnern (siehe Kapitel 3.5).

Tabelle 6 zeigt die Bewertung der für diese Arbeit relevanten Eigenschaften.

	Aussagen-logik	Prädikaten-logik	Semantische Netze	Frames	Beschreibungs-logiken	Ontologien
Klassen, Instanzen, Beziehungen	+	+	+	+	+	+
Mengen, Relationen	-	+	+	+	+	+
Entscheidbar	+	-	-	-	+	+
Schlussfolgerungen	+	+	+	+	+	+
Formale Semantik	+	+	-	-	+	+
Regeln	-	+	-	-	+	+
Restriktionen	-	+	-	-	+	+
Konsistenz	+	+	-	-	+	+
Konzeptualisierung	-	-	-	-	-	+

Tabelle 6: Vergleich der verschiedenen Wissensrepräsentationsformen

Die *Aussagenlogik* scheidet aufgrund der zu geringen Ausdrucksstärke aus. Die *Prädikatenlogik* ist zwar hinreichend ausdrucksstark, jedoch nicht entscheidbar, wodurch sie für einen praxisorientierten Einsatz nicht in Frage kommt. *Semantische Netze* und *Frames* disqualifizieren sich vor allem aufgrund der fehlenden formalen Semantik. Die *Beschreibungslogiken* erfüllen fast alle erforderlichen Kriterien, lediglich eine gemeinsame Konzeptualisierung kann mit der reinen *Beschreibungslogik* nicht umgesetzt werden. Schlussfolgernd zeigt sich, dass **Ontologien** über alle notwendigen Eigenschaften zur Modellierung eines holistischen Ressourcenmodells im Rahmen dieser Arbeit verfügen. Sie sind

- hinreichend ausdrucksstark,
- besitzen eine formale Semantik,
- sind in der Lage, Schlussfolgerungen aus Aussagen zu ziehen und
- garantieren, durch ihre Entscheidbarkeit in einer endlichen Zeit zu terminieren und
- ermöglichen mittels einer gemeinsamen Konzeptualisierung ein gemeinsames Verständnis von Begrifflichkeiten und den Austausch von Informationen zwischen Kommunikationspartnern.

Zur Beschreibung semantischer Zusammenhänge werden bei Ontologien verschiedene Beschreibungslogiken verwendet. Dieser Sachverhalt ist wenig überraschend, da (1) das zu erstellende Ressourcenmodell komplexe Zusammenhänge in einer festen Domäne beschreibt. Dafür wird eine hohe Ausdrucksstärke benötigt. Da das Modell in einem praxisorientierten Proof-of-Concept Softwareframework demonstriert wird, muss die Modellierung entscheidbar sein. Hier zeigen Beschreibungslogiken ihre Stärke. Die Stärke der Ontologien spiegelt sich vor allem in der Interoperabilität zwischen Computersystemen wider. Des Weiteren (2) geht aus verschiedener Literatur und diversen ontologiebasierten Ansätzen (vgl. Kapitel 5.3.3) hervor, dass mit Ontologien praxisrelevante Lösungen modelliert werden können.

Im Folgenden werden Ontologien im Detail erläutert. Über die Jahre wurden unterschiedliche Beschreibungssprachen zu ihrer Modellierung implementiert. Die wichtigsten von ihnen werden ebenfalls im nächsten Kapitel vorgestellt.

5.3 Umsetzung mittels einer Ontologie

Für Ontologien gibt es, je nach Anwendung, eine Reihe von Definitionen. John F. Sowa [84] definiert sie wie folgt:

> *„… An ontology is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language for the purpose of talking about D.“* [84]

Dabei sind zwei Dinge besonders wichtig. Zum einen begrenzt sich der „Katalog“ auf Dinge aus einer ganz bestimmten Domäne. Zum anderen spielt die Perspektive der beschreibenden Person eine essentielle Rolle. Von Peter Karp [85] stammt die Definition, die eine Ontologie mehr hinsichtlich formaler Natur als eine Spezifikation einer Konzeptualisierung beschreibt:

> *„An ontology is a specification of a conceptualization that is designed for reuse across multiple applications and implementations.*
>
> *… a specification of a conceptualization is a written, formal description of a set of concepts and relationships in a domain of interest.”* [85]

Eine Spezifikation einer Konzeptualisierung ist eine geschriebene formale Beschreibung von Konzepten und Beziehungen innerhalb einer bestimmten Domäne. Auch hier ist also die Rede von einer ganz bestimmten Domäne. Die am meisten verbreitete und am häufigsten zitierte Definition stammt von Thomas R. Gruber:

> *“**An ontology is an explicit specification of a conceptualization**. The term is borrowed from philosophy, where an ontology is a systematic account of Existence. For knowledge-based systems, what “exists” is exactly that which can be represented.”* [86]

Eine weitere sehr weit verbreitete Definition, mit Schwerpunkt auf verteilte Systeme bzw. der Mehrfachverwendung von Ontologien stammt von Uschold und Gruninger:

> *„'Ontology' is the term used to refer to the shared understanding of some domain of interest which may be used as a unifying framework to solve the above problems in the above-described manner."* [87]

Fasst man die Definitionen zusammen, lassen sich Ontologien definieren als *„... eine formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung*" [71]. D.h., eine Ontologie ist ein abstraktes Modell (innerhalb einer ganz bestimmten Domäne), in dem alle relevanten Begriffe und Beziehungen identifiziert werden müssen. Dies muss formal erfolgen und damit maschineninterpretierbar sein. Zuletzt muss eine verbindliche Bedeutungsdefinition im Kontext der Anwendung der Ontologie existieren, damit sie von mehreren Parteien genutzt werden kann und Informationen untereinander ausgetauscht werden können.

Dengel fasst in [71] in 5 Punkten die Aufgabe einer Ontologie zusammen:

- *„die Nutzung gemeinsamer Symbole und Begriffe im Sinne einer Syntax,*
- *das gemeinschaftliche (Ein-)Verständnis bezüglich deren Bedeutung, also der Semantik,*
- *die Klassifikation der Begriffe in Form einer Taxonomie,*
- *die Vernetzung der Begriffe mit Hilfe von assoziativen Relationen bei gleichzeitiger*
- *Festlegung von Regeln und Definitionen darüber, welche Relationen sinnvoll und erlaubt sind."*[71]

Was bedeutet es nun, in einer gemeinsamen Sprache zu kommunizieren, welche Bedeutung haben Symbole, Konzepte und Beziehungen? In den Sprachwissenschaften verwendet man hierfür das semiotische Dreieck [88], [89] (nach Ogden und Richards [90]), welches den Zusammenhang zwischen Gedanke (Begriff bzw. Konzept), Symbol und dem Bezugsobjekt (Gegenstand) veranschaulicht. Wird z.B. das Wort „Leiter" ausgesprochen entsteht bei demjenigen, der es hört, ein Bild im Kopf (ein Konzept). Dieses Konzept referenziert den konkreten Gegenstand. Das Problem dabei ist die *Ambiguität* - wenn zwei Zuhörer verschiedene Konzepte von dem Begriff besitzen. Wie im allgemeinen Sprachgebrauch gibt es auch hier mehrere Begriffe, die denselben Kontext beschreiben. Umgekehrt gibt es den Fall, dass einem Begriff verschiedene Bedeutungen (Konzepte) zugeordnet werden können. Im Fall der „Leiter" kann das Homonym das Stufengerät, den Vorgesetzten oder den physikalischen Leiter meinen. Folglich kann es passieren, dass zwei Zuhörer zwar denselben Begriff hören, jedoch unterschiedliche Vorstellungen davon haben.

Verallgemeinert führt dies dazu, dass Systeme mit ähnlichen Zielen verschiedene Darstellungen der Realität besitzen können. Damit zwei Partner (Menschen oder Agenten im Fall von Maschinen) in der Lage sind, im selben Kontext zu kommunizieren, müssen beide ein gemeinsames Konzept über den Gegenstand bzw. den Sachverhalt besitzen. Abbildung 33 veranschaulicht die Vereinbarung eines gemeinsamen Konzeptes im Sinne von Ontologien.

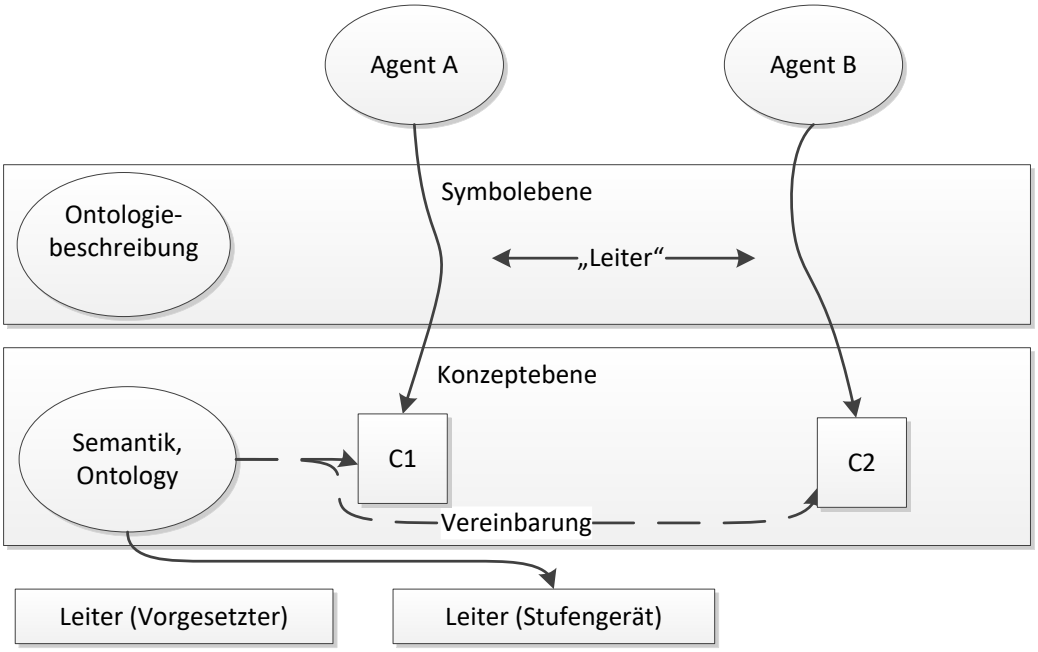


Abbildung 33: Vermittlung von Konzepten

Zwei Agenten tauschen Symbole miteinander aus („Leiter“). Diese Symbole werden auf verschiedenen Konzepten (Leiter im Sinne des Stufengeräts oder des Vorgesetzten) abgebildet. Spezifisches Merkmal von Ontologien ist es nun, zu gewährleisten, dass unterschiedliche Agenten dasselbe Konzept nutzen (können).

Um solch eine Konzeptualisierung informationstechnisch umzusetzen, werden Sprachelemente aus den bereits vorgestellten Formalismen eingesetzt:

- Klassen, Beziehungen und Instanzen: Konzepte sollen durch Klassen repräsentiert werden. Diese werden durch Attribute in Form von Paaren aus dem Namen und Werten beschrieben. Um mit einem konkreten Objekt arbeiten zu können, werden Klassen instanziiert. Instanzen und Klassen können untereinander durch Beziehungen verknüpft werden.
- Relationen: Klassen implizieren, dass die Möglichkeit bestehen muss, Relationen und Mengen abzubilden
- Regeln und Restriktionen: Für Klassen und Beziehungen können Regeln bzw. Restriktionen gelten.
- Axiome: Über Klassen, Beziehungen und Regeln können Aussagen gemacht werden. Man spricht von formalen Axiomen. Axiome bezeichnen Wissen, das durch die übrigen Komponenten nicht hinreichend ausgedrückt werden kann.

In Abbildung 34 sind die verschiedenen Komponenten anhand eines Beispiels visualisiert.

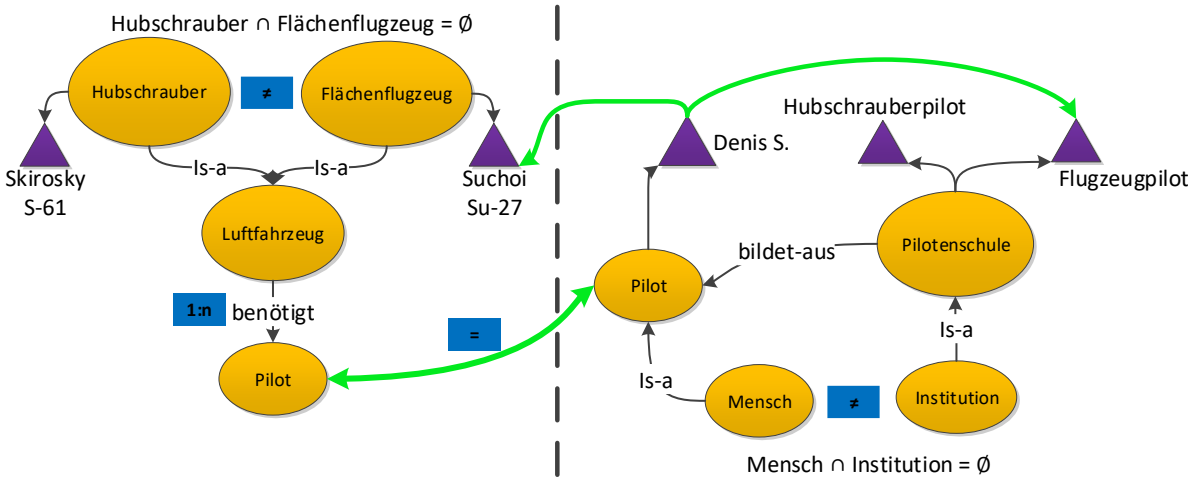


Abbildung 34: Beispiel verschiedener Ontologiekomponenten, die notwendig sind, um ein Konzept hinreichend zu beschreiben

Die Ellipsen repräsentieren unterschiedliche Klassen bzw. Konzepte (*Hubschrauber*, *Flächenflugzeug*, *Luftfahrzeug*, *Pilot*, etc.). Mittels schwarzer Pfeile sind die Beziehungen veranschaulicht. Ein *Hubschrauber* und ein *Flächenflugzeug* sind beides Luftfahrzeuge. Ein *Luftfahrzeug* benötigt einen *Piloten*. Die blauen Rechtecke symbolisieren Regeln. Ein *Luftfahrzeug* braucht demnach mindestens einen *Piloten*. Ein *Hubschrauber* kann nicht gleichzeitig ein *Flächenflugzeug* sein. Das heißt, die Menge von Objekten, die gleichzeitig ein *Hubschrauber* und ein Flächenflugzeug sind, ist leer. Selbiges gilt für die Klassen Mensch und Institution. Die violetten Dreiecke stellen die konkreten *Instanzen* der Klassen dar. Durch die Regel, dass die Klasse *Pilot* auf der linken und rechten Seite dasselbe Konzept repräsentieren, kann nun geschlussfolgert werden, dass die Instanz „Denis S.“, sofern diese mit der Instanz „Flugzeugpilot“ in Verbindung steht, mit der Instanz „Suchoi Su-27“ in Verbindung gebracht werden kann. Anders ausgedrückt bedeutet dies, dass „Denis S.“ ein *Pilot* ist, der eine Ausbildung in der *Pilotenschule* zum „Flugzeugpilot“ absolviert hat und damit in der Lage ist, das *Flächenflugzeug* „Suchoi Su-27“ zu fliegen.

Ontologien erlauben es, Domänenwissen und Anwendungswissen voneinander zu trennen. Auf das Beispiel aus Abbildung 34 angewandt, beschreibt das Domänenwissen den Zusammenhang zwischen Luftfahrzeugen, Pilot und dessen Ausbildung. Anwendungswissen ist das Wissen, das man benötigt, um z.B. abzufragen, wie viele Piloten man für einen Hubschrauber benötigt und wie lange die Ausbildung hierfür dauert. Tauscht man nun das Domänenwissen durch eine andere Ontologie aus, die statt des Luftfahrzeugs Sportgeräte wie Drachen und Gleitschirme beschreibt, ändert sich zwar das Domänenwissen, das Anwendungswissen bleibt jedoch das Gleiche: Man kann die Ontologie nutzen, um abzufragen, wie viele Piloten man für einen Gleitschirm braucht und wie lange die Ausbildung dauert [91].

Ontologien werden auch verwendet, um Domänenwissen zu analysieren und zu validieren [92]. Existiert eine deklarative Spezifikation von Begriffen, kann mit Hilfe einer Ontologie überprüft werden, ob Terme im richtigen Zusammenhang wiederverwendet werden. Zudem können neue

Begriffe bzw. Definitionen eingeführt bzw. bereits bestehende erweitert werden. Dies erweist sich als besonders nützlich, sobald bereits bestehende Ontologien wiederverwendet bzw. erweitert werden.

Obwohl der Modellierungsaufwand trotz Wiederverwendung bereits bestehender Ontologien hoch ist, spricht vieles für den Einsatz und die Entwicklung von Ontologien. In [71] werden die diesbezüglichen Vorteile wie folgt zusammengefasst:

- „*Geringer Aufwand bei Spezifikation und Systemrealisierung durch Wiederverwendung und ontologiegestützt strukturierte Kommunikation*
- *Höhere Zuverlässigkeit der Modellierung (die Ontologien sind gewachsen und auf breiter Front akzeptiert)*
- *Höhere Flexibilität der Lösung, da etwa Details von Konzepten erst zur Laufzeit nachgeschlagen werden, also beim Entwurf noch nicht bekannt sein müssen*
- *Bessere Nutzerfreundlichkeit, z.B. durch Unterstützung von Suchfunktionen (...),*
    - *bei der Wartung (...) und*
    - *bei der Erhebung von Informationen (durch einfaches Klassifizieren).“*

Weiterführende Informationen zu Ontologien im Allgemeinen, sowohl aus der Sicht der Philosophie als auch der Informatik, finden sich in [80] und [71] .

5.3.1 Ontologietypen

In der Informatik unterscheidet man zwischen vier Typen (vgl. Abbildung 35) von Ontologien [93].

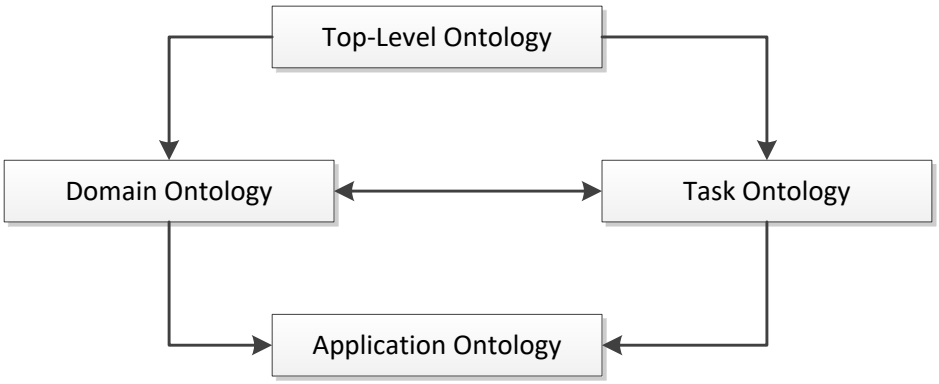


Abbildung 35: Die vier Typen von Ontologien nach [93]

(1) *Top-Level Ontologien* (auch als *Upper Ontology* bekannt) beschreiben generelle Konzepte im Allgemeinen, ohne dabei auf eine bestimmte Domäne oder Problemstellung einzugehen.

(2) *Domain Ontologien* beinhalten Konzepte, die einer bestimmten Domäne oder Problemstellung zugeordnet werden können. Ein Beispiel wäre eine Ontologie, die Luftfahrzeuge beschreibt, diese in verschiedene Typen und Kategorien unterteilt und Gemeinsamkeiten bzw. Unterschiede aufzeigt.

(3) *Task Ontologien* beschreiben grundlegende Konzepte, die sich auf eine allgemeine Aktivität oder Aufgabe beziehen. Der Fokus liegt dabei auf „allgemein“, d.h. keine speziellen Aufgaben. Möchte man z.B. einen generellen Prozess beschreiben, sind *Task Ontologien* das Mittel der Wahl.
(4) *Application Ontologien* beschreiben ganz bestimmte Aufgaben bzw. Prozesse. Verbindet man die *Domain* mit der *Task Ontologien*, führt dies zu den *Application Ontologien*. Hierbei fokussiert man sich auf eine spezielle Aufgabe in einer ganz speziellen Domäne.

Eine weitere Möglichkeit, Ontologien zu unterscheiden, bietet sich anhand ihrer Ausdrucksstärke. In Abbildung 36 ist das Ontologiespektrum nach Lassila und McGuiness [94] dargestellt.

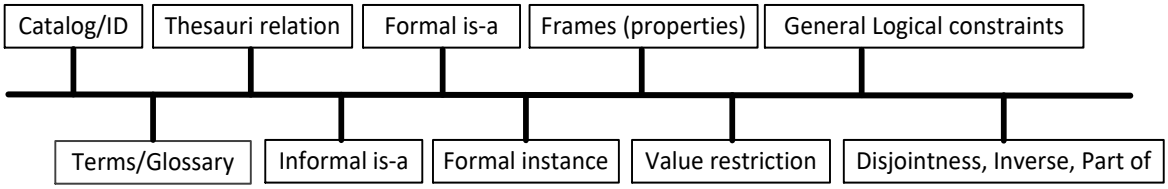


Abbildung 36: Ontologiespektrum nach Lassila und McGuiness [94]

Das Spektrum ist von links (leichtgewichtige Ontologien, ausdrucksschwach) nach rechts (schwergewichtige Ontologien, ausdrucksstark) sortiert. Erwartungsgemäß schließen (leichtgewichtige) Ontologien die bereits diskutierten Formalismen wie z.B. *Frames* mit ein. Ganz links sind Ontologien mit einem kontrollierbaren Vokabular (*Catalog/ID*), auf der rechten Seite solche, die komplexe Beschreibungslogiken verwenden, um Wissen zu beschreiben. An dieser Stelle soll nicht zu sehr auf die einzelnen Kategorien eingegangen, sondern nur ein kleiner Überblick gegeben werden. Details können in [94] nachgelesen werden.

- *Catalog/ID*: Es steht (nur) eine endliche Liste von Begriffen (Wörterbuch) zu Verfügung, die genutzt werden können, um einen Sachstand zu beschreiben.
- *Glossary*: Es existiert ein Wörterbuch mit Begriffen, die genutzt werden können. Zusätzlich ist die Bedeutung der Begriffe in natürlicher Sprache formuliert.
- *Thesauri relation*: Es existiert ein kontrolliertes Vokabular, die Bedeutung der Begriffe ist bekannt, und zusätzlich stehen die Begriffe durch Relationen in gegenseitigem Bezug. Ein Thesaurus besitzt Synonyme, Hierarchien, Homonyme und Assoziationen.
- *Informale is-a*: Ab dieser Ausdrucksstärke spricht man von *Taxonomiesystemen*. Dinge werden hier in Gruppen bzw. in Hierarchien angeordnet. Eine informale is-a Hierarchie besitzt eine explizite Hierarchie von Klassen, jedoch kann eine Instanz unterschiedlichen Klassen zugeordnet werden.
- *Formal is-a*: Hier spricht man von *strikten Hierarchien*, d.h. eine Instanz kann immer nur zu einer Klasse gehören. Alle Klassen sind disjunkt.
- *Formal instance*: Bei formalen Instanzen sind Beziehungen nicht nur zwischen Klassen erlaubt, sondern ebenfalls zwischen Instanzen.
- *Frames*: In dieser Kategorie inkludieren Klassen Informationen von Eigenschaften. Diese werden ebenfalls an die Subklassen und Instanzen vererbt.

- *Value restriction*: Noch mehr Ausdrucksstärke bieten Ontologien dieser Kategorie. Wie der Name schon sagt, besteht hier die Möglichkeit, Eigenschaften mit Restriktionen zu versehen. Zum Beispiel kann festgelegt werden, dass eine Eigenschaft nur Zahlenwerte annehmen kann oder die Zahlenwerte in einem vorgegebenen Bereich liegen müssen.

Aus Abbildung 36 wird ersichtlich, dass es eine Reihe von unterschiedlichen Ontologietypen mit speziellen Charakteristiken gibt. Je komplexer die zu beschreibenden Zusammenhänge sind, desto ausdrucksstärker muss die Ontologie sein. Das heißt, es werden mehr Sprachkonstrukte zur Beschreibung benötigt (vgl. Kapitel 5.1.5). Während ein einfacher Klassenbaum zur Kategorisierung von Luftfahrzeugen mittels einer einfachen Taxonomie beschrieben werden kann, benötigt z.B. ein Ressourcenmodell, wie es im Rahmen dieser Arbeit vorgestellt wird, eine Reihe ausdrucksstarker Sprachelemente, damit die Anforderungen aus Kapitel 1.4 mittels des Ressourcenmodells erfüllt werden können.

5.3.2 Beschreibungssprachen

Aus dem vorigen Kapitel ist bekannt, dass zur Modellierung des im Rahmen dieser Arbeit entwickelten Ressourcenmodells Beschreibungslogiken die notwendigen Sprachkonstrukte bereitstellen. Als nächstes wird eine maschineninterpretierbare Beschreibungssprache benötigt, die die Beschreibungslogik umsetzt und damit das Manipulieren, Repräsentieren und Speichern von Daten und Metadaten ermöglicht. Diese wird genutzt, um das Ressourcenmodel in Software zu implementieren.

Im Rahmen dieser Arbeit verwendet (u.a. wegen breiter Werkzeugunterstützung) und aufgrund des etablierten Einsatzes im *Semantic Webs* unter dem *World Wide Web Consortium* (W3C) werden die Sprachen RDF (und RDFS) und dessen Erweiterung OWL bzw. OWL2 vorgestellt. Außerdem werden SWRL, eine regelbasierte Erweiterung von OWL, und die Abfragesprache SPARQL vorgestellt.

5.3.2.1 RDF und RDF-Schema (RDFS)

Das *Ressource Description Framework* (RDF) ist eine formale Sprache, die vom W3C im Jahr 1999 erstmals veröffentlicht wurde. RDF dient der Wissensrepräsentation und Beschreibung strukturierter Informationen (im Web). Sie ermöglicht es, Daten in einem Netzwerk auszutauschen, zu verarbeiten und zu kombinieren. Dabei mussten nach [71] folgende Gesichtspunkte bei der Konzipierung beachtet werden:

- *„Wissen ist (im Web) verteilt. Es sind geeignete Identifizierungs- und Zugriffsmechanismen notwendig.*
- *Wissen (im Web) ist subjektiv. Aussagen müssen diskutierbar sein.*
- *Es gibt viele unterschiedliche Interessengruppen. Die Repräsentationsformen für Wissen müssen demnach entsprechend erweiterbar und einfach zu benutzen sein.“*

Der ursprüngliche Fokus der Sprache lag auf der Modellierung von Metadaten, also Daten, die einen bestimmten Datensatz beschreiben. 2004 wurde die Spezifikation erweitert, um eine Darstellung semantischer Informationen zu ermöglichen, die über das Web hinausgehen. Ziel war es, eine XML-Untermenge mit fester Semantik festzulegen, sodass das bereits maschinenlesbare XML maschinenverstehbar (Informationen können korrekt interpretiert werden) wird.

RDF-Dokumente basieren auf einer (gerichteten) Graphen-orientierten Darstellung. Dies ermöglicht es, Aussagen über Ressourcen (zu beschreibende Dinge bzw. Objekte), sogenannte *Statements* (RDF-Tripel: *Ressource*, *Property*, *Value*), zu machen. Kanten und Knoten werden durch eindeutige Bezeichner (URIs) identifiziert. Die Beschreibung der Daten wird mittels *Properties* (Eigenschaften zur Beschreibung der Ressource) realisiert. *Uniform Resource Identifier* (URI) sind eine Verallgemeinerung von URLs (Web-Adressen), um Objekte eindeutig zu identifizieren. Obwohl die Objekte nicht zwingend online verfügbar sein müssen, hat man sich dazu entschlossen, das Konstruktionsschema der URLs beizubehalten. So könnte das Objekt „Flugzeug“ mit der URI

„http://www.beispielontologie.de/Flugzeug“

beschrieben werden. Dabei muss die URL nicht existieren. URIs können abstrakte Ressourcen beschreiben, mit denen eine Maschine nicht direkt etwas anfangen kann; sie sind eine Referenz auf die zu modellierenden Entitäten (vgl. Abbildung 37). Ein Vorteil der Graphen ist ihre dezentrale Anordnung.

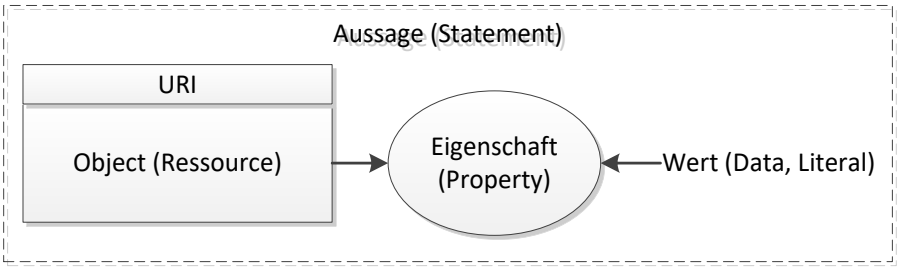


Abbildung 37: Bestandteile eines *Statements*, bestehend aus der *Ressource*, den *Properties*, der *URI* und einem Wert, der einer Eigenschaft zugeordnet wird.

Mit RDF ist es möglich, Informationen aus verschiedenen Quellen zu kombinieren.

```
1  <Flugzeug>
2  <Typ> Eurofighter </Typ>
3  <Pilot> Denis S. </Pilot>
4  </Flugzeug>

5  <Pilot>
6  <Name> Denis S. </Name>
7  <Flugzeug> Eurofighter </Flugzeug>
8  </Pilot>

9  <Flugzeug Pilot="Denis S" Typ="Eurofighter"/>
```

Listing 1: Unterschiedliche Darstellungsarten derselben Information

Diese Kombination von Quellen kann durch Schlussfolgerungsmechanismen wiederum neue interessante und relevante Informationen aufzeigen. RDF definiert also ein Datenmodell zur Beschreibung der Semantik von Daten und erlaubt Assoziationen zwischen Objekten bzw. Ressourcen. Das in Listing 1 dargestellte Beispiel soll die Notwendigkeit einer einheitlichen Repräsentationsform demonstrieren.

Zusehen ist dieselbe Information, dargestellt in drei verschiedenen XML-Varianten. In allen drei Beispielen (markiert durch die unterschiedlichen Farben) wird modelliert, dass Denis S. ein Eurofighter Pilot ist. Das Problem besteht darin, dass, obwohl es sich jeweils um dieselbe Information handelt, diese auf verschiedene Arten repräsentiert wird (unter Verwendung derselben Sprache). Für einen Menschen stellt es keine allzu große Schwierigkeit dar, die Information in allen drei Varianten auf dieselbe Art zu interpretieren. Die Information in Listing 1 kann auch mittels einer graphischen Notation dargestellt werden (vgl. Abbildung 38).

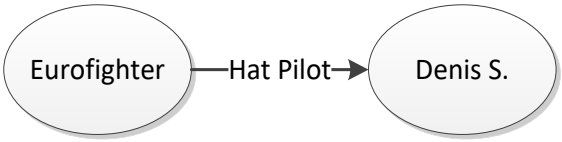


Abbildung 38: Darstellung von Informationen mittels einer graphischen Notation

Die Darstellung von Informationen in einem Graphen ist übersichtlich und gut lesbar, (solange die Graphen nicht zu komplex werden), jedoch für eine Verarbeitung durch eine Maschine weniger gut geeignet. Deswegen werden Informationen in einer RDF-Beschreibung durch *Tripel* dargestellt. Man spricht auch von einer Subjekt-Prädikat-Objekt Darstellung. Ein Beispiel für ein Tripel wäre

„Eurofighter“(S) – „hat Pilot“(P) – „Denis S.“(O)

Um die Tripel in einem Dokument abbilden und von einer Maschine bzw. einem Parser interpretieren lassen zu können, gibt es verschiedene Syntaxarten.

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#""
   xmlns:bsp="http://www.bsp.de/Flugzeug#">
3
4  <rdf:Description rdf:about="http://www.bsp.de/Eurofighter">
5      <bsp:hatPilot rdf:datatype="http://www.w3c.org/2001/XMLSchema#string”>
6          Denis S.
7      </bsp:hatPilot>
8  </rdf:Description>
9
10 <rdf:Description rdf:about="http://www.bsp.de/Eurofighter">
11     <bsp:hatTyp rdf:resource="Kampfflugzeug" />
12 </rdf:Description>
13 </rdf:RDF>
```

Listing 2: Beispiel für ein RDF Dokument mit der XML Syntax zur Beschreibung des Objekts "Eurofighter“

Zwei verbreitete Syntaxarten sind die N-Tripel und die Turtle Darstellung. Beide sind aus der von Tim Berners-Lee vorgeschlagenen *Notation 3*[23] entstanden.

Die mit Abstand am häufigsten eingesetzte Syntax benutzt jedoch den XML Standard. Dies liegt unter anderem daran, dass der XML Standard bereits etabliert ist und sehr viele Werkzeuge, Bibliotheken für verschiedene Programmiersprachen und Parser für XML-Dokumente zur Verfügung stehen. In Listing 2 ist der Inhalt eines RDF-Dokumentes zur Beschreibung des Objektes Eurofighter in der XML Syntax zusehen.

In den Zeilen 2 und 3 werden Namensräume für den RDF-Namensraum und den eigenen Namensraum festgelegt (*rdf* und *bsp*), um eine kürzere Schreibweise und bessere Lesbarkeit zu ermöglichen. In den Zeilen 5 bis 11 wird das Objekt `http://www.bsp.de/Eurofighter` beschrieben. Das Schlüsselwort dafür ist in RDF `rdf:Description.` Das zu beschreibende Objekt steht in dem Attribut `rdf:about="…".` In Zeile 6 wird das Literal „Denis S." beschrieben, es wird also auf einen einfachen Datentyp wie einen *String* oder einen Zahlenwert verwiesen. Die Beschreibung des Datentyps wird auf das XML Schema referenziert. In Zeile 10 wird mit dem Schlüsselwort `rdf:resource="…"` auf eine andere Ressource verwiesen.

Die RDF-Eigenschaften können wie folgt zusammengefasst werden:

- Unabhängigkeit: Jedes *Property* ist eine Ressource, folglich können beliebige *Properties* erfunden werden.
- Austauschbarkeit: RDF basiert auf XML und damit auf einem Standard, der leicht kommuniziert werden kann.
- Skalierbarkeit: Jedes *Statement* besteht aus einem Tripel und ist damit einfach und performant von einer Maschine verarbeitbar.
- Werte können Ressourcen sein: Der Wert eines Tripels kann ein Literal oder wieder eine Ressource sein. Damit kann der Wert auch wieder ein *Property* besitzen.
- *Statements* können Ressourcen sein (*RDF-Reification*): *Statements* können selbst auch wieder Ressourcen sein und damit ebenfalls wieder *Properties* besitzen.

Ein RDF-Model ist also eine Menge von Statements. Jedes Statement entspricht einem Tripel aus Subjekt, Prädikat und Objekt. Ein Subjekt und ein Prädikat sind Ressourcen. Ein Objekt ist entweder eine Ressource oder ein Literal. Jedes relationale Datenmodell ist in eine RDF-Tripel Darstellung überführbar.

RDF Schema

Während RDF dazu genutzt wird, Aussagen über Ressourcen (Beschreibung von Entitäten, Beschreibung von Beziehungen zwischen zwei Entitäten und das Zuweisen von Typen) und deren Eigenschaften zu treffen, wird mit *RDF Schema* (RDFS), eigentlich *RDF Vocabulary Descrip-*

[23] Die Notation3 (N3) war zu komplex weshalb die W3C Recommendation den weniger komplexen Teil der N3 Notation extrahiert hat und es unter dem Namen N-Tripels veröffentlicht hat. N-Tripels wurden später zu der heute unter dem Namen Turtle bekannten Syntax erweitert.

tion Language, die Bedeutung der Daten modelliert. RDFS ist damit eine Beschreibungssprache zur Definition strukturierter Vokabularien. Hierzu spezifiziert RDFS terminologisches Wissen und dessen Zusammenhänge in Form von Klassen- und Beziehungshierarchien (z.B. Subklassen). Mit RDFS können weiterhin Eigenschaften und Restriktionen von Beziehungen festgelegt werden. Hierzu führt RDFS erweiterte Sprachkonstrukte ein (z.B. `rdfs:domain`, `rdfs:range`, `rdfs:Class`, etc.). RDFS ist selbst wieder ein gültiges RDF-Dokument, wodurch RDFS eine Erweiterung von RDF ist.

Für einen Menschen ist klar, dass der Eurofighter eine Bezeichnung für ein Kampfflugzeug ist, die Eigenschaft „fliegen“ besitzt und einen Piloten, der wiederum ein Mensch ist, benötigt, der es steuert. Für eine Maschine sind das zunächst nur Symbole ohne eine festgelegte Bedeutung. Der Maschine muss der Zusammenhang zwischen z.B. Flugzeug, Pilot und Mensch erst bekannt gemacht werden, damit diese die Schlussfolgerungen eines Menschen nachahmen bzw. die Informationen auf eine ähnliche Weise interpretieren kann. RDFS ermöglicht es, dieses terminologische Wissen (Schemawissen) zu spezifizieren. Anders ausgedrückt, stellt RDFS die Funktion zur Verfügung, Aussagen über semantische Beziehungen innerhalb eines vorgegebenen Vokabulars zu machen. Durch die Spezifikation eines solchen Schemawissens wird RDFS zu einer Ontologiesprache, mit der viele (wenn auch nicht alle, siehe OWL) semantische Abhängigkeiten innerhalb einer Domäne beschrieben werden und somit „leichtgewichtige“ Ontologien erstellt werden können. Eine detailliertere Beschreibung mit allen Details beinhaltet das Buch „Semantic Web“ von Pascal Hitzler et al.[41].

5.3.2.2 OWL und OWL 2

RDF und RDFS fehlt es an der notwendigen Ausdrucksstärke. Zum Beispiel können lokale Domain- und Range Restriktionen nicht beschrieben werden. Es fehlen außerdem Existenz- und Kardinalitätsrestriktionen. Weiterhin fehlen Eigenschaften der Transitivität, Symmetrie und Inversen. Die folgenden Sätze aus der natürlichen Sprache können in RDFS nicht hinreichend genau beschrieben werden:

- Jede Perzeptionskette besteht aus mindestens einem Bildverarbeitungsmodul
- Module der Perzeptionskette mit der Eigenschaft XYZ sind immer genau Modul A und Modul B.
- Die Vorbedingung einer Ressource A gilt auch für Ressource B, wenn diese mit Ressource A verknüpft ist

Um unter anderem mit diesen Restriktionen umgehen zu können, wurde die Beschreibungssprache *Web Ontology Language* OWL[24][41] als Erweiterung von RDF und RDFS vom W3C Konsortium im Jahre 2004 vorgestellt. OWL ist eine Beschreibungslogik mit wohldefinierter Semantik, die auf der Prädikatenlogik erster Ordnung basiert. Sie ermöglicht weiterhin das Schlussfolgern, um auf implizites Wissen schließen zu können. Eine OWL Ontologie besteht,

[24] Zum Thema „Warum OWL und nicht WOL“, eine E-Mail-Korrespondenz: http://lists.w3.org/Archives/Public/www-webont-wg/2001Dec/0169.html

wie bereits kennengelernt, aus Klassen (Konzepten), Rollen (engl. *Properties,* Beziehungen zwischen Klassen bzw. Instanzen) und Individuen (Instanzen von Klassen). Man unterscheidet zwischenabstrakten (*objecttype properties*) und konkreten (*datatype properties*) Rollen. . OWL folgt der *Open World Assumption*, d.h. wird ein bestimmter Umstand nicht modelliert, bedeutet es nicht, dass er nicht existiert. In OWL gibt es keine *Unique Name Assumption*, d.h. es ist möglich, ein und dasselbe Objekt mit verschiedenen Namen zu adressieren. Verschiedenheit muss dementsprechend explizit ausgedrückt werden. Umgekehrt ist es möglich, zwei unterschiedliche Objekte mit demselben Namen zu adressieren.

Um mit OWL-Dokumenten arbeiten zu können, sind im Laufe der Zeit verschiedene Werkzeuge, Bibliotheken und Editoren entstanden. Eine der populärsten, welche auch im Rahmen dieser Arbeit eingesetzt wird, ist Protegé [95], ursprünglich entwickelt von der Stanford Universität.

Es gibt verschiedene Arten von OWL-Syntax: RDF(S)-Syntax, abstrakte Syntax und DL-Syntax. Listing 3 zeigt ein simples Beispiel für die drei genannten Arten.

```
1  <owl:Class rdf:ID="Eurofighter">
2      <rdfs:subClassOf rdf:resource="#Flugzeug"/>          RDF(S)- Syntax
3  </owl:Class>

4  Class(a:Eurofighter partial a:Flugzeug)                  abstrakte Syntax

5  Eurofighter ⊆ Flugzeug                                   DL-Syntax
```

Listing 3: Verschiedene OWL-Syntax Arten

Die am weitesten verbreitete (aufgrund von verschiedenen Werkzeugen) und im Rahmen dieser Arbeit relevante Syntax-Art ist die RDF(S)-Syntax.

Ein OWL-Beispiel zur Modellierung der Disjunktion von Klassen (die Klasse „Sensor“ und „Platform“ sind disjunkt) ist in Listing 4 dargestellt: mittels Inferenz kann geschlossen werden, dass „UAV“ und „Sensor“ ebenfalls disjunkte Klassen sind. Das Schlüsselwort `rdfs:subClassOf` wird genutzt, um Klassenhierarchien aufzubauen. „UAV“ ist damit eine Subklasse von „Platform“. Das Schlüsselwort `owl:equivalentClass` drückt Äquivalenzen zwischen verschiedenen Klassen aus.

```
1  <owl:Class rdf:ID="UAV">
2      <rdfs:subClassOf rdf:resource="#Platform"/>
3  </owl:Class>

4  <owl:Class rdf:ID="Sensor">
5      <owl:equivalentClass rdf:resource="#Camera"/>
6      <rdfs:subClassOf rdf:resource="#Ressource"/>
7  </owl:Class>

8  <owl:Class rdf:ID="Platform">
9      <owl:disjointWith rdf:resource="Sensor"/>
10 </owl:Class>
```

Listing 4: Definition von disjunkten Klassen

Weiterhin können inverse Rollen beschrieben werden („requires capability“ – „capability required by“). Da keine *Unique Name Assumption* besteht, gibt es die Möglichkeit, die Verschiedenheit von Individuen auszudrücken. Es können komplexe Klassenbeschreibungen modelliert werden (z.B. gehört ein Individuum zur Klasse „UAV Sensor“, wenn es zur Klasse „Sensor“ gehört und die Rollenzuweisung „*equippedTo* UAV“ besitzt).

Wie bereits angesprochen, gibt es zwei verschiedene Arten von Rollen. Abstrakte Rollen beschreiben Beziehungen zwischen Klassen (z.B. Platform *equippedWith* Sensor), konkrete Rollen beschreiben Daten und Eigenschaften von Klassen bzw. Instanzen, in dem am Ende eines Tripels ein Wert und kein Objekt steht (vgl. Listing 5).

```
1  <owl:NamedIndividual rdf:about="Tau 640">
2      <rdf:type rdf:resource="FLIR Sensor"/>
3      <nimmtWahr rdf:resource="Wärmesignatur"/>
4      <gewicht rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3</gewicht>
5  </owl:NamedIndividual>
```

Listing 5: Beispiel für eine Klassenzuordnung einer Instanz und die Verwendung von einem *Dataproperty*

Das Individuum „Tau 640“ wird der Klasse „FLIR Sensor“ zugeordnet (Zeile2). In der nächsten Zeile ist das *Objectproperty* „nimmtWahr“ zu sehen, welches auf die Klasse „Wärmesignatur“ verweist. In Zeile 4 wird festgelegt, dass dieses Individuum das *Dataproperty* „gewicht“ vom Typ Integer mit dem Wert 3 besitzt. Zusätzlich können Rollen sogenannte *Domain* und *Range* Restriktionen besitzen, die den Klassenraum für eine Rolle beschränken (vgl. Listing 6).

```
1  <owl:ObjectProperty rdf:ID="equipped">
2      <rdfs:domain rdf:resource="Sensor"/>
3      <rdfs:range  rdf:resource="Platform"/>
4  </owl:ObjectProperty>

5  <owl:DataProperty rdf:ID="gewicht">
6      <rdfs:domain rdf:resource="Sensor"/>
7      <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
8  </owl:DataProperty>
```

Listing 6: Beispiel für das *Objectproperty* "fliegt" mit dazugehöriger Domain- und Range Restriktion

In dem genannten Beispiel wurde z.B. festgelegt, dass die Rolle „equipped“ nur zwischen den Klassen „Sensor“ und „Platform“ definiert werden darf. Die *Domain* und *Range* Restriktionen gelten auch für *Dataproperties*, jedoch wird dort als *Domain* die Klasse festgelegt und als *Range* der Datentyp.

Um komplexe Klassen aus einfachen Klassen aufbauen zu können, gibt es in OWL verschiedene Schlüsselwörter, die den Einsatz von logischen Konstruktoren erlauben. Diese sollen der vollständigkeitshalber an dieser Stelle erwähnt, jedoch nicht weiter im Detail diskutiert werden. Zu den Konstruktoren zählen

- Logisches Und (Konjunktion): `owl:intersectionOf`
- Logisches Oder (Disjunktion): `owl:unionOf`
- Logische Negation: `owl:complementof`

Um komplexe Klassen durch Rollen beschreiben zu können gibt es ebenfalls verschiedene Werterestriktionen (`owl:hasValue`, `owl:allValuesFrom`, `owl:someValuesFrom`) und Kardinalitätsrestriktionen (`owl:cardinality`, `owl:minCardinality`, `owl:maxCardinality`).

Für Rollen existieren ebenfalls verschiedene Schlüsselwörter. Über `owl:subPropertyOf` können Rollenhierarchien aufgebaut werden. Inverse Rollen werden über das Schlüsselwort `owl:inverseOf` definiert. Neben inversen Rollen können Rollen ebenfalls als Identisch (`owl:equivalentProperty`) beschrieben werden. Schließlich gibt es eine Reihe von Rolleneigenschaften, die im Folgenden näher erläutert werden sollen.

1. **Symmetrie**: Eine symmetrische Rollenbeziehung besagt, dass, wenn A ein Teil von B ist, dann B auch ein Teil von A ist.

$$r(a,b) \rightarrow r(b,a)$$

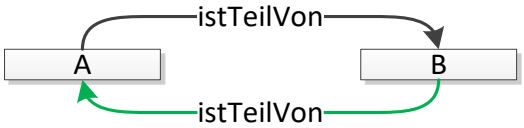


Abbildung 39: Symmetrische Rollenbeziehung

2. **Transitivität**: Wenn A ein Teil von B ist und B ein Teil von C, dann ist auch A ein Teil von C.

$$r(a,b), r(b,c) \rightarrow r(a,c)$$

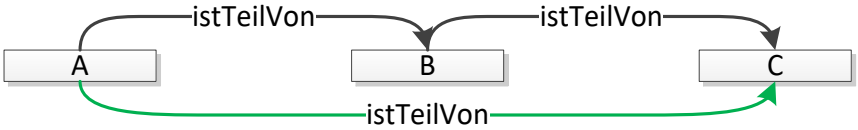


Abbildung 40: Transitive Rollenbeziehung

3. **Funktionalität**: Wird eine Rolle als funktional deklariert und verweist sie auf zwei verschiedene Individuen B und C, dann bedeutet dies, dass B und C im Sinne von `owl:sameAs` gleich sind.

$$r(a,b), r(a,c) \rightarrow b = c$$

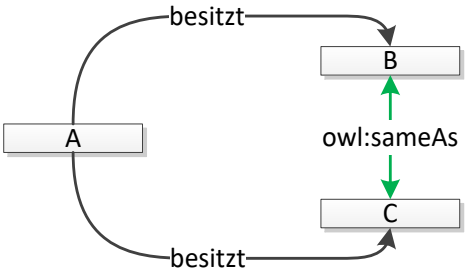


Abbildung 41: Funktionale Rollenbeziehung

4. **Reflexivität (OWL 2)**: Man spricht von einer reflexiven Rolle, wenn diese eine Beziehung zum Individuum selbst herstellt. Das Gegenteil ist eine irreflexive Rollenbeziehung.

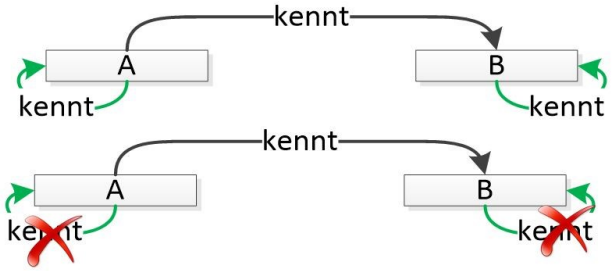


Abbildung 42 Reflexive und irreflexive Rollenbeziehung

Weiterhin kann jeder Rolle ihre **Inverse** zugeteilt werden (vgl. Abbildung 43).

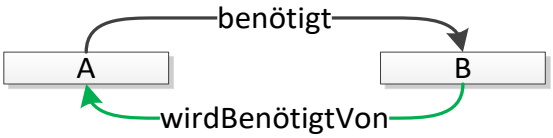


Abbildung 43: Inverse Rollenbeziehung

OWL erlaubt es, Informationen sowohl aus der TBox als auch aus der ABox abzufragen, aus der TBox z.B. Klassenäquivalenzen, Subklassenbeziehungen, Disjunktion von Klassen und Klassenkonsistenzen. Abfragen, die an die ABox gerichtet sind, können genutzt werden, um zu überprüfen, ob ein Individuum zu einer bestimmten Klasse gehört. Es können alle Individuen einer Klasse abgefragt werden. Über die Rollenbeziehungen kann abgefragt werden, in welcher Relation zwei Individuen zu einander stehen.

OWL Varianten

OWL ist ein semantisches Fragment der Beschreibungslogik, die wiederum eine Untermenge der Prädikatenlogik ist. Aus Kapitel 5.1.5 ist bekannt, dass eine Reihe unterschiedlich ausdrucksstarker Beschreibungslogiken (vgl. Tabelle 5) mit verschiedenen Eigenschaften existieren. In OWL spiegelt sich das durch drei unterschiedliche Ausführungen, die sich in ihrer Ausdrucksstärke und Komplexität unterscheiden, wider:

$$OWL\ Lite\ \subseteq OWL\ DL\ \subseteq OWL\ Full$$

Dabei ist jede Variante eine Teilmenge der nächsthöheren Variante. OWL Lite ist damit vollständig in OWL DL (*Description Logic*) und OWL DL vollständig in OWL Full enthalten. Das in Kapitel 5.3.2.1 vorgestellte RDFS ist ein vollständiges Fragment von OWL Full, nicht jedoch von OWL DL (es sind nur ausgewählte RDFS Sprachelemente erlaubt) bzw. OWL Lite. RDFS besitzt Sprachelemente, die dazu beitragen, dass eine Ontologie nicht entscheidbar ist (Stichpunkt: Reinfikation, Klassen können als Instanzen interpretiert werden, etc.). Da sowohl OWL

Lite als auch OWL DL entscheidbar sind, OWL Full jedoch nicht, ist dies auch der Grund, warum das vollständige RDFS in OWL Full, jedoch nicht in den anderen Sprachvarianten enthalten ist. Abbildung 44 zeigt einen Übersichtsgraphen der verschiedenen OWL Varianten.

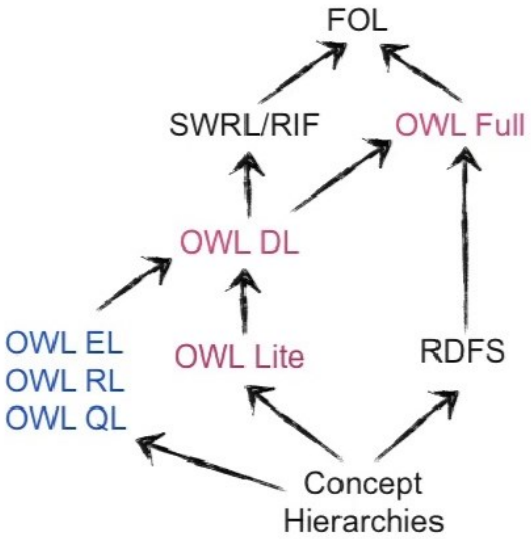


Abbildung 44: OWL Hierarchie[25]

Wie bereits erwähnt, ist OWL ein Fragment der Prädikatenlogik (FOL). Für OWL DL existieren weiterhin das *Rule Interchange Format* (RIF) und die *Semantic Web Rule Language* (SWRL), welche die Ausdrucksstärke von OWL DL mittels Regeln erweitern. In OWL 2 existieren weitere leichtgewichtige OWL Varianten (OWL EL, RL und QL). Weiterführende Informationen zu OWL EL, RL und QL finden sich in [97] .

Die kleinste Variante von OWL ist **OWL Lite**. Lite bietet einfach zu implementierende Sprachfragmente an. Sie nutzt die Beschreibungslogik $\mathcal{SHIF}(\mathcal{D})$(vgl. Tabelle 5). OWL Lite besitzt z.B. keine Aufzählungsklassen, keine Kardinalitäten bzw. Kardinalitätsrestriktionen (erlaubt ist nur die Kardinalität 1 und 0) und kann keine Disjunktion ausdrücken. Weiterhin gibt es Beschränkungen, die verhindern, komplexe Klassen zu beschreiben (z.B. fehlt `unionOf, complementOf, disjointWith`, etc.) Die Laufzeitkomplexität für die Entscheidbarkeit ist exponentiell. In der Praxis zeigt sich, dass OWL Lite zu ausdrucksschwach ist, weshalb meistens OWL DL eingesetzt wird.

OWL DL nutzt die Beschreibungslogik $\mathcal{SHOIN}(\mathcal{D})$ und besitzt eine formal definierte Semantik die sich auf die in Kapitel 5.1.5 vorgestellte Beschreibungslogik $\mathcal{ALC}$ zurückführen lässt. Jedoch lässt sich mit $\mathcal{ALC}$ nicht die gesamte Ausdrucksstärke von OWL beschreiben, weshalb die Beschreibungslogik auf $\mathcal{SHOIN}(\mathcal{D})$ erweitert werden muss. OWL DL ist entscheidbar, vollständig und korrekt. Die Laufzeitkomplexität für die Entscheidbarkeit ist nicht deterministisch exponentiell, jedoch für die meisten praktischen Anwendungen noch ausreichend. OWL DL verwendet nur explizite RDFS Sprachelemente, damit die Entscheidbarkeit gewahrt wird.

[25] Bildquelle: https://tinyurl.com/ycqbtssl, 10.05.2018

Klassen und Rollen müssen beispielsweise explizit getrennt werden. Konkrete Rollen, also *Datatype Properties*, dürfen weder als transitiv, symmetrisch, invers oder invers funktional deklariert werden. OWL DL ist die in der Praxis am häufigsten eingesetzte Variante von OWL und hat eine sehr gute Unterstützung durch Entwicklungswerkzeuge und Inferenzmaschinen.

OWL Full vermischt die OWL Syntax mit RDFS, d.h. mit dieser Variante ist man in der Lage, Metamodellierungen und Reinfikation durchzuführen. Da OWL Full konzipiert wurde, um ausdrucksstarkes Wissen repräsentieren zu können, enthält die Semantik Aspekte, die aus logischer Sicht problematisch sein können.

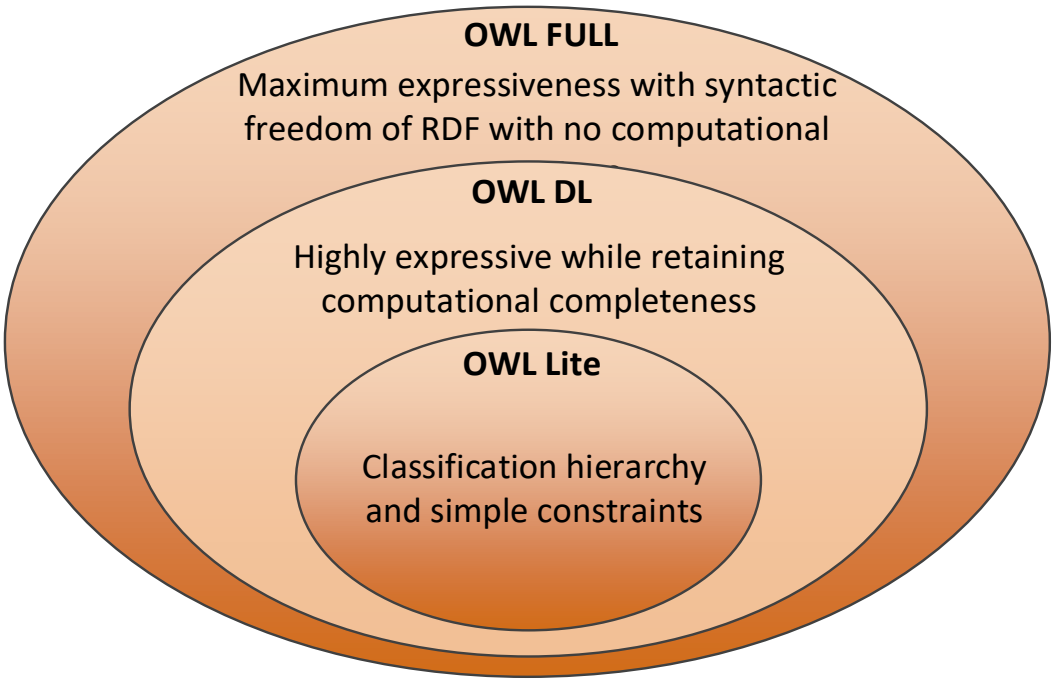


Abbildung 45: Die drei OWL Varianten: OWL Full, OWL DL, OWL Lite[26]

Hier ist die uneingeschränkte Nutzung aller OWL und RDFS-Sprachelemente möglich. Daraus ergibt sich, dass Typisierungen nicht mehr zwingend eingehalten werden müssen, was wiederum dazu führt, dass z.B. Klassen gleichzeitig Instanzen sein können. Damit ist z.B. `owl:Thing` dasselbe wie `rdfs:resource` und `owl:Class` dasselbe wie `rdfs:Class`. `Owl:DatatypeProperty` ist eine Subklasse von `owl:ObjectProperty`. Letzten Endes führt dies dann dazu, dass OWL Full nicht entscheidbar ist. OWL Full wird verwendet, wenn hohe Ausdrucksstärke wichtiger ist als die Garantie einer Entscheidbarkeit. Diese Variante hat wenig Praxisrelevanz, da es nur wenig Werkzeuge (Entwicklungswerkzeuge und Inferenzmaschinen) und Bibliotheken gibt, die sie unterstützen. Abbildung 45 veranschaulicht die drei OWL-Varianten und ihre jeweils wichtigsten Eigenschaften.

[26] https://tinyurl.com/ycv5wj9n

OWL 2

Man hat festgestellt, dass es in der Praxis Fälle gibt, die mit OWL nicht oder nur sehr schwer beschrieben werden können. Aus diesem Grund hat man sich dazu entschlossen, diese Schwachstellen mit einer neuen OWL Version, OWL 2 (*W3C Recommendation* seit Oktober 2009 [98]), zu beseitigen und die Beschreibungslogik zu erweitern. Dabei wurde darauf geachtet, die Kompatibilität mit OWL weitgehend zu wahren. Während OWL die Beschreibungslogik $\mathcal{SHOIN}(\mathcal{D})$ nutzt, verwendet OWL 2 die ausdrucksstärkere Logik $\mathcal{SHROIQ}(\mathcal{D})$. Vergleicht man die Buchstaben mit Tabelle 5, stellt man fest, dass bis auf R alle Buchstaben bekannt sind. R steht für RBox. Die TBox und ABox wurden bereits vorgestellt. Bisher stand das Wissen über Klassen und Relationen in der TBox. Hier wird nun dieses Wissen so aufgeteilt, dass die TBox weiterhin das Wissen über Klassen enthält, die Relationen (Rollen) jedoch nun in der RBox abgelegt werden.

In OWL existieren drei unterschiedliche Profile (OWL Lite, OWL DL, OWL Full), die sich in ihrer Ausdrucksstärke unterscheiden. Diese wurden in OWL 2 angepasst. OWL 2 Full ist eine Erweiterung von OWL Full, jedoch mit den zusätzlichen Axiomen, die durch $\mathcal{SHROIQ}(\mathcal{D})$ bereitgestellt werden. Aufgrund der niedrigen Relevanz wurde OWL Lite entfernt und durch drei neue Profile ersetzt. Der Fokus der drei neuen Profile liegt auf einer effizienten Berechnung (in Polynomialzeit berechenbar) (vgl. Abbildung 44) [99]. Bei den drei Profilen handelt es sich um

- OWL 2 EL,
- OWL 2 QL (Querry Language),
- OWL 2 RL (Rule Language).

OWL 2 EL ist die simpelste OWL-Variante und nutzt die Beschreibungslogik $\mathcal{EL}++$. Zu dieser zählen die Konjunktion, existentielle Restriktionen, die *Top* ($\top$) und *Bottom* ($\bot$) Klasse, abgeschlossene Klassen, eingeschränkte Rollen, allgemeine Rolleninklusion und die Transitivität. Der Vorteil liegt neben der einfachen Implementierung vor allem in der Laufzeit der Komplexität zur Berechnung der Entscheidbarkeit. Zudem gibt es eine Reihe von Ontologien, die auf Leistung ausgelegt sind und bereits mittels der simpelsten OWL-Variante beschrieben werden können [100]. OWL 2 QL basiert auf der Beschreibungslogik DL Lite und wurde entwickelt, um schnell und bequem Schnittstellen zwischen Datenbanken und dem *Semantic Web* entwickeln zu können und es auf großen Datensätzen anzuwenden. OWL 2 QL unterstützt Oberklassen, Unterklassen, inverse Rollen und einfache Rollenhierarchien. Die ABox orientiert sich an $\mathcal{SHROIQ}(\mathcal{D})$. OWL 2 RL basiert auf einem sogenannten Horn-Regel-Fragment [101] von OWL 2 und ermöglicht es, Subklassenaxiome als Regeln zu interpretieren. Es weist damit eine Ähnlichkeit zu Regelsprachen auf. Ziel dieser Variante ist es, regelbasierte Systeme in die Welt des *Semantic Webs* miteinzubeziehen. Details zu OWL 2 sind [98] zu entnehmen.

5.3.2.3 Regelbasierte Sprache: SWRL

Um OWL mehr Ausdrucksstärke und Flexibilität zu verleihen, erlaubt OWL die Kombination einer regelbasierten Sprache, der *Semantic Web Rule Language* (SWRL) [102] mit der RuleML. SWRL kann somit als eine Kombination aus OWL DL und RuleML bezeichnet werden und lässt sich damit ebenfalls durch die Beschreibungslogik $\mathcal{SHOIN}(D)$ beschreiben. SWRL wurde 2004 von der W3C als Member Submission eingereicht und mittlerweile in viele OWL-Systeme integriert, unter anderem im Protegé Editor und dem Pellet Reasoner. SWRL definiert Regeln wie folgt:

$$a_1 \wedge a_2 \ \wedge \ldots \wedge a_n \ \rightarrow \ b_1 \wedge b_2 \ \wedge \ldots \wedge b_n$$

Die Atome a_i beschreiben die Vorbedingungen und bilden den Rumpf einer Regel. b_i wird als Kopf (oft auch Konklusion) bezeichnet und bildet die Nachbedingungen. Bei den Atomen kann es sich sowohl um Klassen ($C(x)$) bzw. einen Klassenausdruck C handeln, als auch um Properties ($P(x, y)$), also Beziehungen. Weiterhin können die Atome folgende Funktionen annehmen: `sameAs(x,y)`, `differentFrom(x,y)` und `builtin`$(r, z_{1,\ldots,z_n})$. Die Regeln sind zu lesen als:

„*wenn Vorbedingung X wahr ist, gilt auch die Nachbedingung*“.

Eine leere Vorbedingung ist immer wahr, eine leere Nachbedingung immer falsch. Alle Regeln, die mit OWL-Axiomen ausgedrückt werden können, können mit SWRL Regeln abgebildet werden. Umgekehrt gibt es SWRL-Regeln, die nicht mit OWL-Axiomen ausgedrückt werden können. Zusätzlich enthält SWRL eine Reihe von *Builtins:* Funktionen, unterteilt in verschiedene Kategorien, die das Erstellen von Regeln erleichtern: Vergleichs-Builtins, mathematische Builtins, String-Builtins, Builtins zur Datums- und Zeitbehandlung, etc. Einige von ihnen sind die folgenden: *swrl:equal*, *swrl:add*, *swrl:stringEqualIgnoreCase*, *swrl:dateTime*. Ein Beispiel für eine SWRL-Regel ist das Folgende: (Variablen werden in SWRL immer mit einem vorangestellten „?“ versehen.)

$$Flugzeug(?f1) \wedge Pilot(?p1) \wedge hasPilot(?f1, ?p1) \rightarrow Flugbereit(?f1)$$

Die Regel besagt, dass, wenn *f1* zur Klasse Flugzeug gehört und *p1* zur Klasse Pilot, und das Flugzeug *f1* mit dem Piloten *p1* über die Beziehung *hasPilot* verknüpft ist, folglich das Flugzeug *f1* als flugbereit klassifiziert wird (alle Instanzen von *f1* der Klasse „Flugbereit“ zugeordnet werden).

5.3.2.4 Abfragesprache: SPARQL

Die Abfragesprache, um RDF Graphen zu traversieren, ist die *SPARQL Protocol and RDF Query Language* (SPARQL[27]). SPARQL ist an SQL, eine Abfragesprache für relationale Datenbanken, angelehnt und seit 2008 ein W3C Standard. Das Übertragungsprotokoll und das Rückgabeformat (XML basiert) werden direkt von SPARQL festgelegt. Mit SPARQL ist es

[27] Der Name SPARQL ist ein rekursives Akronym

möglich, Werte aus einem RDF Graphen zu extrahieren und die Daten wiederum als URIs, typisierte bzw. nicht-typisierte Literale oder wiederum als RDF Subgraphen auszugeben. Weiterhin können damit unbekannte Beziehungsstrukturen innerhalb der Daten identifiziert werden. Im Falle von verteilten Datenbanken können mittels der Abfragesprache in einer einfachen Abfrage komplexe Joint-Operationen, wie man sie aus SQL kennt, durchgeführt werden. Weiterhin ist die Transformation von RDF-Daten von einem Vokabular in ein anderes möglich.

SPPARQL Variablen werden an RDF Terme gebunden und beginnen immer mit einem „?“. Die Abfrage wird wie in SQL über das SELECT Statement durchgeführt. Das Ergebnis wird in Form einer Tabelle angezeigt. Abfragen werden in Form von Tripel beschrieben, wobei in dem Tripel an beliebiger Stelle eine oder mehrere Variablen stehen können. Das nachfolgende Beispiel liefert alle Instanzen, die mit der Rolle *equipped* verbunden sind:

```
1  PREFIX bsp: <http://beispiel.de/#>
2  SELECT ?Sensor, ?Platform
3      WHERE {
4          ?Sensor bsp:equipped ?Platform .
5      }
```

Listing 7: Beispiel für eine SPARQL Abfrage mit zwei Variablen

Da SPAQRL auf der bereits erwähnten Turtle Serialisierung basiert, muss jede Abfrage mit einem Punkt (.) beendet werden. `?Sensor` und `?Platform` sind zwei Variablen innerhalb der Abfrage. Mit dem SELECT Statement werden die Variablen festgelegt. Mit WHERE legt man die Tripel fest, die abgefragt werden sollen. Weitere Statements sind z.B. PREFIX (zum Anlegen von Abkürzungen für Namensbereich, um die Lesbarkeit zu erhöhen), FROM (zum Angeben einer oder mehrerer konkreter Ontologien bzw. RDF-Quellgraphen), BASE, ORDER, etc. Das Ergebnis würde wie folgt aussehen:

?Sensor	?Platform
Sony FCB	DJI
Flir Tau	DJI

Tabelle 7: Beispiel für das Ergebnis einer SPARQL Abfrage

Möchte man mehrere Abfragen miteinander verknüpfen, schreibt man alle Abfragen untereinander. Diese werden automatisch mittels einer UND-Verknüpfung miteinander verknüpft (vgl. Listing 8). Möchte man eine ODER-Verknüpfung zwischen zwei Abfragen benutzten muss dies explizit mit dem UNION-Statement angegeben werden.

```
1  SELECT ?Sensor, ?Capability, ?Platform
2      WHERE{
3          ? Sensor bsp:equipped ?Platform .
4          ? Sensor bsp:providesCapability ?Capability.
5      }
```

Listing 8: SQL Abfrage bestehend aus zwei Abfragen

Hier wird nun nicht nur nach den Sensoren und den Plattformen gesucht, sondern ebenfalls nach den Fähigkeiten der Sensoren (z.B. „sense thermal energy“). Diese Art der Abfrage kann

ebenfalls dazu genutzt werden, um die Ergebnisse einzuschränken bzw. nach bestimmten Werten zu filtern.

```
1  SELECT ?Sensor, ?weight, ?Platform
2      WHERE{
3          ? Sensor bsp:equipped ?Platform.
4          ? Sensor bsp:providesCapability bsp:senseThermalEnergy.
5          ? Sensor bsp:hasWeight ?weight.
6          FILTER (?weight < 3) }
```

Listing 9: Erweitertes SPQRAL Beispiel

In dem Beispiel aus Listing 9 wird das Ergebnis dahingehend eingeschränkt, dass nur nach Sensoren mit der Fähigkeit „sense thermal energy“ (Zeile 4) gesucht wird und die außerdem ein Gewicht von weniger als 3 (die Einheit wird in diesem Beispiel nicht berücksichtigt) haben (Zeile 6 mittels des FILTER Statements).

Neben der SELECT-Abfrage gibt es noch folgende Möglichkeiten, eine Abfrage zu schreiben:

- ASK: liefert boolesche Antworten („gibt es ein Ergebnis für eine Abfrage?“)
- CONSTRUCT: erstellt neue RDF-Tripel, indem Variablen gebunden werden
- DESCRIBE: liefert Server-generierte RDF Beschreibungen von Abfragen

5.3.3 Ontologiebeispiele zur Ressourcenmodellierung

In Kapitel 5.1 wurden unterschiedliche Formalismen zur Wissensrepräsentation vorgestellt. Aus Kapitel 5.1.6 ist bekannt, dass im Bereich der Computerwissenschaften Ontologien verbreitet sind, um Informationen und deren Verknüpfungen aus unterschiedlichen Domänen maschinenverarbeitbar und interpretierbar zu modellieren. In Kapitel 5.3.2 ging es um die am häufigsten eingesetzten Beschreibungssprachen zur Modellierung von Ontologien. In diesem Kapitel sollen abschließend Arbeiten vorgestellt werden, die auf die dargestellten Technologien zurückgreifen und Ähnlichkeiten zur Themenstellung dieser Arbeit aufweisen.

Durch die weite Verbreitung der ontologiebasierten Wissensrepräsentation sind verschiedene *Design Pattern* [103] entstanden, die es vereinfachen, Ontologien zu modellieren und Informationen in einer standardisierten Art und Weise zu speichern.

Die Literaturrecherche ergab, dass es nur wenige Arbeiten gibt, die sich unmittelbar auf den vorliegenden Anwendungsfall, dem Modellieren von Ressourcen und deren Fähigkeiten im Allgemeinen und im spezifischen von technischen Systemen und bildverarbeitender Daten-Prozessierung an Bord von Luftfahrzeugen, beschäftigen. Bisherige Arbeiten befassten sich entweder ausschließlich mit der Modellierung der Sensorik des Luftfahrzeugs oder mit der fliegenden Plattform selbst. Grundsätzlich sind diese ermutigenden Ansätze jedoch für diese Arbeit nicht ausreichend. Vergleicht man die Arbeiten mit der Aufgabenstellung aus Kapitel 1.4 und den Anforderungen aus Kapitel 3, lässt sich feststellen, dass die Notwendigkeit für ein holistisches Ressourcenmodel im Vordergrund steht, welches das gesamte Luftfahrzeug mitsamt aller missionsrelevanten Systemkomponenten beschreibt.

Am wichtigsten erweist sich die Veröffentlichung von Mario Gomez et al. [104],[105]. Die Autoren beschreiben ihr Problem als

> *„knowledge-based matchmaking: matching the ISR (intelligence, surveillance and reconnaissance) requirements of task to the ISR-providing capabilities of available sensors "*

an Bord von z.B. UAVs oder anderen mobilen Plattformen. Um Aufgaben und Charakteristika von militärischen Missionen zu analysieren, nutzen sie das *Mission and Means Framework* (MMF), welches methodische Ansätze zur Entscheidungsfindung bei Überwachungsaufgaben im militärischen Kontext zur Verfügung stellt. In ihrer Arbeit repräsentieren sie das MMF als Ontologie. Besonders interessant ist die Tatsache, dass Gomez et al. die einzelnen eingesetzten Aufklärungssensoren mit der eigentlichen sensortragenden Plattform durch die Beschreibung der Mission und spezifischen Aufgaben in Verbindung setzt. In dem so genannten *Matchmaking* Prozess werden die im Rahmen der Mission benötigten Fähigkeiten mit denen der zur Verfügung gestellten Sensoren verglichen. Anschließend konnte so bestimmt werden, welche Sensoren an Bord der Plattform vorhanden sein müssen, um eine bestimmte Mission bzw. Aufgabe durchführen zu können. Inferenzmechanismen analysieren, welche Sensoren bzw. Plattformen für eine bestimmte Missionsaufgabe zur Verfügung stehen bzw. eingesetzt werden können. Obwohl bereits externe Ontologien wie z.B. OntoSensor [106] , *CIMA* [107] und *MMI (Marine Metadata Interoperability Project) Platform Ontology* [108] für die Modellierung von Sensoren und Aufgaben existierten, können diese nicht die Anforderungen von Gomez gänzlich erfüllen. Letztlich erstellten Gomez et al. eine eigene Ontologie. Diese wurde ergänzt mit einer angepassten Taxonomie zur Beschreibung von benötigen und vorhandenen Fähigkeiten (vgl. Abbildung 46).

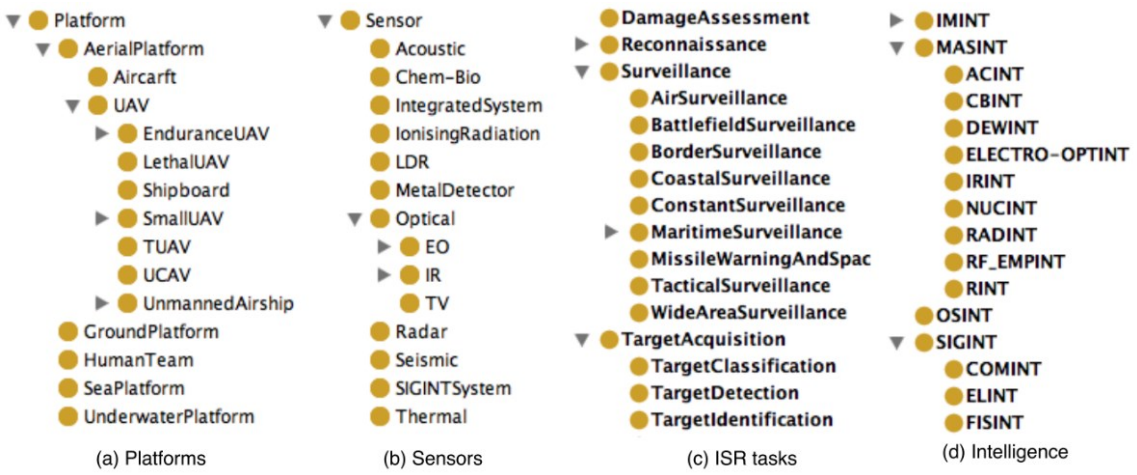


Abbildung 46: Die von M. Gomez et al. verwendete Klassenhierarchie [104]

Die von Gomez et al. aufgestellt Taxonomie ist sehr breit und beschreibt ein größeres Anwendungsfeld als jenes, welches im Rahmen dieser Arbeit betrachtet wird. Während hier z.B. als Plattform nur UAVs betrachtet werden und der Schwerpunkt auf optischen Sensoren liegt, betrachten M Gomez et al. zum einen ebenfalls Bodenfahrzeuge, Schiffe, Unterwasserfahrzeuge

und menschliche Einsatzkräfte, sowie neben den optischen außerdem akustische, chemische und einige weitere Sensoren.

Obwohl dieser Ansatz im Bereich der Sensor-, Missions- und Plattformmodellierung vielversprechend ist, fehlt bei dem Konzept die Betrachtung perzeptiver Fähigkeiten und der datenverarbeitenden Algorithmen. Weitere Ressourcen, wie z.B. die Sensorsteuerung, Datenlink und Stromversorgung, bleiben ebenfalls unberücksichtigt. Gomez et al. konzentrieren sich auf den Betrachtungspunkt eines Missionskommandanten, während der Fokus dieser Arbeit auch auf einer Anwendung der Ontologie an Bord des Luftfahrzeugs während einer Mission im Rahmen eines automatisierten Systems liegt. Darüber, wie die Sensoren verknüpft werden, welche Algorithmen aktiviert oder welche anderen Nebenbedingungen beachtet werden müssen, wird bei Gomez et al. keine Aussage gemacht.

Nichtsdestotrotz zeigt dieser Ansatz die Anwendbarkeit und den Mehrwert bereits vorhandener Ontologien und wie Frameworks und Standards sinnvoll in semantischer Art und Weise wiederverwendet werden können. Da sowohl Sensoren als auch perzeptive Fähigkeiten bzw. Bildverarbeitungsalgorithmen im Rahmen dieser Arbeit eine besondere Rolle spielen, sollen im Folgenden bereits bestehende Ontologien aus diesen Bereichen beschrieben werden.

5.3.3.1 Sensormodellierung

Aus dem Bereich der Sensormodellierung existieren zwei bekannte Ontologien, die näher betrachtet werden sollen. Bei der Ersten, *OntoSensor* [106], handelt es sich um eine Wissensdatenbank, vergleichbar mit den Strukturen des Semantischen Webs. Die Ontologie beinhaltet

> *„… definitions of concepts and properties adopted in part from SensorML, extensions to IEEE SUMO and references to ISO 19115. SensorML is developed as a specification for efficient implementation by vendors that desire to implement OGC compliant sensor systems.“* [106]

SUMO steht für *IEEE Suggested Upper Merged Ontology* und definiert allgemeine Terme, die in vielen verschiedenen Domänen allgemeingebräuchlich sind. *OntoSensor* nutzt die Top-Level Ontologie von SUMO und erweitert diese. Mit der *OntoSensor* Ontologie wird das Ziel verfolgt, Daten von verschiedenen Sensoren, wie z.B. akustische Sensoren und Infrarotsensoren, zu fusionieren, um mit den gewonnenen Informationen feindliche Taktiken und Strategien besser einschätzen und bewerten zu können. Das Modellieren der Sensoren innerhalb der Ontologie konzentriert sich auf die Charakteristika und Funktionalitäten der Sensoren, jedoch nicht auf ihre abstrahierten perzeptiven *High-Level* Fähigkeiten (z.B. „Wahrnehmen von Wärmesignaturen“), wie es im Rahmen dieser Arbeit notwendig ist. Zudem gab es seit dem Jahre 2006 keine Veränderungen und Erweiterungen mehr an der *OntoSensor* Ontologie.

Eine Alternative zu *OntoSensor* stellt die *Semantic Sensor Network* (SSN) Ontologie dar, publiziert von der *W3C SSN Incubator Group* (SSN-XG). Die SSN-XG beschreibt ihre Ontologie wie folgt:

> *"(The ontology) can describe sensors in terms of capabilities, measurement processes, observations and deployments."* [110]

Die SSN Ontologie wurde erstellt, indem bereits existierende Sensor-Ontologien und Standards [110] untersucht wurden, um Anwendungsfälle aus den folgenden Kategorien abzudecken:

- *Data discovery and linking*
- *Device discovery and selection*
- *Provenance and diagnosis*
- *Device operation, tasking and programming*

Insgesamt untersuchte die Gruppe zwölf unterschiedliche Ontologien, mit dem Ergebnis, dass keine der bestehenden alle Anwendungsfälle abdecken konnte. Aus diesem Grund wurde eine neue Ontologie erstellt, welche sensorspezifisch und kompatibel mit dem OGC Standard ist. Weiterhin wurden die bekanntesten und am weitesten verbreiteten Definitionen und Konzepte mit in die Ontologie aufgenommen, um so anderen Domänenexperten zu ermöglichen, diese zu erweitern. Zusätzlich wurde die SSN Ontologie an das *DOLCE-UltraLite* (DUL) Konzept angepasst. Mit DOLCE (*Descriptive Ontology for Linguistic and Cognitive Engineering*) wird eine weitere Top-Level Ontologie bezeichnet, die von der *WonderWeb Foundation Ontologies Library* (WFOL) entwickelt wurde.

> "*It is intended to act as a starting point for comparing and elucidating the relationships with other ontologies of the library and also for clarifying the hidden assumptions underlying existing ontologies or linguistic resources...*" [40]

Wie bereits Gomez et al. feststellten, besteht bei der SSN Ontologie das gleiche Problem wie auch z.B. bei der *OntoSensor* Ontologie: Beide sind nicht dafür ausgelegt, High Level Fähigkeiten von Ressourcen (z.B. von Sensoren oder von der Flugplattform) auf aufgabenorientierte Anforderungen abzustimmen. Im Rahmen dieser Arbeit wurden Teile der SSN Ontologie wiederverwendet und mit Konzepten, die an die Ontologie von Gomez et al. angelehnt sind, erweitert.

5.3.3.2 Algorithmenmodellierung zur Bildverarbeitung

Der Fokus der zu entwickelnden Ontologie liegt jedoch nicht nur auf physikalischen Entitäten wie Luftfahrzeugen oder Sensoren, sondern auch auf daten- und bildverarbeitenden Elementen. Insofern sollen auch bestehende Ontologien, die sich mit Computer Vision und Bildverarbeitung beschäftigen, betrachtet werden.

A. Jungmann und B. Kleinjohann stellen in [111] einen Ansatz vor, um das Service-orientierte Paradigma des *Semantischen Webs* auf die bildverarbeitende Domäne zu übertragen. Dies soll es ermöglichen, automatisiert Bildverarbeitungsapplikationen zu generieren. Ein Service kapselt dabei eine bestimmte Funktionalität (im Sinne des Problems, welches mit einem speziellen Datenverarbeitungsalgorithmus korreliert). Die verschiedenen Services können anschließend flexibel miteinander kombiniert werden, um eine komplexere Funktionalität bereit zu stellen.

Jungmann und Kleinjohann unterscheiden zwischen elementaren und zusammengestellten Services. Elementare Services sind *Black Boxes,* welche durch ihren Eingang und Ausgang beschrieben werden. Ein zusammengestellter Service ist eine Kombination aus mehreren elementaren Services. Um eine wissensbasierte Spezifikation für bildverarbeitende Funktionalitäten bereitzustellen, entwickeln sie Basiskonzepte und das dazu korrespondierende Vokabular.

> "*The ontology is then used as basis for formally specifying the functionality provided by services as well as the required functionality in terms of a request."* [111]

Die Spezifikation beachtet dabei die Eingangsdaten (z.B. ein Bild), die zu erwartenden Ausgangsdaten (z.B. ROIs) und die durchzuführende Aufgabe. Eine Aufgabe kann aus Teilaufgaben bestehen, bei denen der Ausgang einer Teilaufgabe den Eingang einer anderen Teilaufgabe darstellt. Die Beschreibung der Eingangs- und Ausgangsdaten wird verwendet, um festzustellen, welche elementaren bzw. zusammengestellten Services miteinander verknüpft werden können. Die Veröffentlichung von Jungmann und Kleinjohann konzentriert sich nur auf Bildverarbeitung. Die Eingangs- und Ausgangsdaten werden mittels einer Ontologie beschrieben, um das Service Paradigma des *Semantic Webs* zu realisieren. Damit deckt ihre Arbeit nur ein Subset des in dieser Arbeit beschriebenen Problems ab (vgl. Kapitel 1.3).

Einen ähnlichen Ansatz stellen Regis Clouard et al. in [112], [113] vor. Sie beschreiben ein wissensbasiertes System, dass das Generieren von bildverarbeitenden Applikationen automatisiert und ausführbare Programme zur Verfügung stellt.

> "*Users describe tasks to perform on images and the system constructs a specific plan, which, after being executed, should yield the desired results.*"

Um dieses Ziel zu erreichen, konstruieren sie Bildverarbeitungsketten unter der Berücksichtigung von Auswahl, Parameterabstimmung und Ablaufplanung bereits vorhandener Bildverarbeitungsalgorithmen. Die Autoren heben hervor, dass der Grundgedanke nicht darauf abzielt, das Design von Bildverarbeitungsalgorithmen zu beschreiben, sondern vielmehr bestehende Algorithmen so zu klassifizieren, dass bestimmt werden kann, für welche Anwendungsbereiche und Aufgaben sie eingesetzt werden können. Um automatisiert Ketten von Operationen, unter Berücksichtigung der Mehrdeutigkeit von Bildinformationen, der Ungenauigkeit von Verarbeitungsschritten und empirischen Analysemethoden generieren zu können, schlagen sie zwei Lösungen vor: Die Erste betiteln sie als *knowledge-rich problem-solving* und die Zweite ist hierarchischer Natur. Während die hierarchische Lösung im Rahmen dieser Arbeit nicht weiter relevant ist, beinhaltet die erste Lösung interessante Konzepte. Regis Clouard et al. stellen drei Ebenen vor, um Daten zu beschreiben und zu verarbeiten: die *physikalische* Ebene (z.B. Größe, Sensortyp, natürliches Rauschen), die *perzeptive* Ebene (z.B. Punktwolken, Kontrast, Texturregionen) sowie die *semantische* Ebene (z.B. kleinste Objektgröße, Form des Objekts, topologische Beziehungen).

Regis Clouard et al. [113] stellt dabei ein Ontologie-basiertes Modell vor, um bildverarbeitende Anwendungen abzubilden. Die Autoren konzentrieren sich auf Informationen, die notwendig

sind, um eine bildverarbeitende Software zu designen und zu evaluieren. Als Implementierungssprache für ihre Domänenontologie verwenden sie OWL. Durch die semantische Formulierung von standardisierten Konzepten und Termen ist es sowohl für Bildverarbeitungsexperten, als auch automatisierten Computer Vision Systemen möglich, das Modell zu verstehen. Obwohl die Anforderungen in dieser Arbeit mit denen von Regis Clouard et al. korrelieren, ist die Ontologie nicht für abstrakte Szeneninterpretationen auf einer High Level Ebene geeignet, sondern beschreibt die Bildverarbeitung auf einem sehr niedrigen Level. Sie begründen die Notwendigkeit einer solchen Beschreibung unter anderem damit, dass ein Bild zunächst keine relevanten Informationen enthält. Erst wenn man weiß, wonach man sucht, können relevante Informationen von irrelevanten unterschieden werden:

> „*Therefore, one can conclude that the development of image processing applications is not an activity that operates without purpose and knowledge about the objective. The reason is twofold. First, image acquisition is responsible for creating incomplete, degraded, and corrupted images that under constrain the observed scene (e.g., noise, blur, loss of motion or depth). Second, the image content has no meaning in itself. An image is inherently ambiguous and does not provide information about its content. Without a goal, an image cannot allow to discriminate between relevant and irrelevant information*." [114]

Regis Clouard et al. beschreiben, welche Konzepte in Anbetracht der drei Level (physikalisch, perzeptiv und semantisch) identifiziert werden können.

Aus dem Bereich der Robotik diskutiert G. Lim et al. [115] einen Ontologie-basierten Ansatz zur Wissensrepräsentation von Service-Robotern. Sie unterscheiden zwischen Low-Level Sensor- und Motordaten und semantischen High-Level Informationen. Ziel ist es, ein Modell zu entwickeln, welches von einem Roboter benutzt werden kann, um dessen Umgebung wahrnehmen und die ihm zugewiesenen Aufgaben verstehen zu können. Weiterhin muss der Roboter in der Lage sein, mit einem Menschen zu interagieren und zu verstehen, welche Absicht dieser beim Stellen einer Aufgabe verfolgt. Als Beispiel nennen sie die Aufgabe „Bring mir meine Tasse." Um die Aufgabe bewältigen zu können, muss der Roboter z.B. wissen, was eine Tasse ist und wo sich diese befindet. Weiterhin müssen Low-level Fähigkeiten des Roboters angesprochen werden, wie z.B. die Bewegung zur Tasse (Motorik), das Erkennen der Tasse (Mustererkennung) und das Greifen der Tasse (Motorik). Infolgedessen wird ein holistisches Model benötigt, das die Aufgabenbeschreibung, die daraus resultierenden Aktionen und welche Komponenten des Rotors zur Durchführung angesprochen werden müssen, beinhaltet.

5.3.3.3 Fazit

Aus den gezeigten Beispielen kann geschlussfolgert werden, dass sowohl für die Sensormodellierung als auch die Algorithmenmodellierung Ansätze vorhanden sind, die eine Ähnlichkeit mit Teilaspekten der Aufgabenstellung dieser Arbeit aufweisen. Es zeigt sich aber auch, dass es an Ansätzen fehlt, die die gesamte Aufgabenstellung dieser Arbeit bedienen können. Mario

Gomez et al. stellen ein Konzept vor, das die meisten Überschneidungen mit dieser Arbeit aufweist, aber auch sie kommen zu dem Schluss, dass es keine Ontologie gibt, die sie 1:1 übernehmen können, sodass sie gezwungen waren, eine eigene Ontologie zu erstellen.

Dies trifft auch für den Kontext dieser Arbeit zu. Die vorgestellten Beispiele enthalten Ideen und Ansätze, die zwar relevant sind, jedoch nicht vollständig übernommen werden können. Regis Clouard et al. beschreiben z.B. einen Modellierungsansatz für Bildverarbeitungsalgorithmen, der für diese Arbeit zwar interessant ist, aber eine zu tiefe Modellierungstiefe beinhaltet. Dennoch kann die Arbeit von Regis Clouard et al. als eine mögliche detaillierte Erweiterung gesehen werden, die zu einem späteren Zeitpunkt in die im Rahmen dieser Arbeit entstandene Ontologie eingefügt werden kann, um eine erweiterte Beschreibung von bildverarbeitenden Algorithmen zu erhalten.

Um die Aufgabenstellung samt Anforderungen erfüllen zu können, wird im Rahmen dieser Arbeit eine Ontologie modelliert, mit der alle notwendigen Konzepte beschrieben werden können. Um die Interoperabilität zu erhöhen, werden so weit wie möglich Verknüpfungen zu bereits bestehenden Ontologien gezogen. Eine davon ist z.B. die vorgestellte SSN Ontologie.

Die Gründe für die Wahl dieser Alternative werden im Folgenden zusammengefasst:

- Die SSN Ontologie wird in vielen anderen Applikationen eingesetzt, wodurch sich die Interoperabilität zwischen verschiedenen Domänen und Anwendungen erhöht.
- Von den zwei bekanntesten Sensorontologien (*OntoSensor* und SSN) scheint die *OntoSensor* Ontologie weder gepflegt noch erweitert zu werden. Damit ist SSN die Ontologie die aktuellste und vollständigste Sensor Ontologie.
- Die SSN Ontologie baut auf einer weitverbreiteten Top-Level Ontologie (DUL) auf und enthält damit Konzepte aus unterschiedlichen Domänen. Dies wiederum erlaubt es, die im Rahmen dieser Arbeit entwickelte Ontologie in einem breiteren Kontext zu verwenden und die neu eingeführten spezifischen Konzepte einer größeren Zahl an Ontologie-Experten, sofern erforderlich, zur Verfügung zu stellen.

Im nächsten Kapitel wird darauf eingegangen, wie die genannten Konzepte und Methoden für ein ablauffähiges Gesamtsystem entworfen, implementiert, integriert und anschließend bewertet werden.

6

Implementierung

Im Folgenden wird auf die Implementierung der Konzepterweiterung, wie sie in Kapitel 4 beschrieben wurde, in Kombination mit den methodischen Festlegungen zum Ressourcenmodell aus Kapitel 5, eingegangen. In Kapitel 4.2 wurden die wichtigsten Komponenten der Erweiterung (*Perception Solver, Processing Coordinator* und dem *Resource Manager*) kurz vorgestellt. Nun wird die Funktionsweise und Implementierung dieser Komponenten detailliert erläutert. Der Schwerpunkt liegt auf der Ontologieumsetzung und Integration in das bestehende S&PM Framework. Zunächst werden jedoch der Kommunikationsaufbau und der Datenaustausch vorgestellt.

6.1 Modulkommunikation und -synchronisation

Aus der Konzepterweiterung in Kapitel 4 ist bereits bekannt, dass die Softwarearchitektur einen modularen Aufbau verfolgt. Jede Softwarekomponente ist als eigener Softwareprozess implementiert. Zwischen den Prozessen findet eine Synchronisierung statt. Konkret bedeutet das, dass der *Perception Solver*, *Processing Coordinator*, *Resource Manager* und andere Softwarekomponenten innerhalb des Konzepts unabhängig voneinander gestartet, gestoppt und daraus resultierend verändert werden können.

Für den Datenaustausch und die Synchronisation der Daten zwischen den Prozessen wurden Schnittstellen für eine Interprozesskommunikation implementiert und geeignete Datenformate definiert. Insbesondere bei (Absturz und) Neustart einzelner Prozesse ist es notwendig, durch entsprechenden initialen Datenaustausch konsistente Datenstände in den einzelnen Prozessen sicherzustellen.

Im Folgenden soll der Kommunikationsablauf vom Initialisieren der einzelnen Komponenten, über die Synchronisation untereinander, bis hin zum Start einer Perzeptionskette erläutert werden.

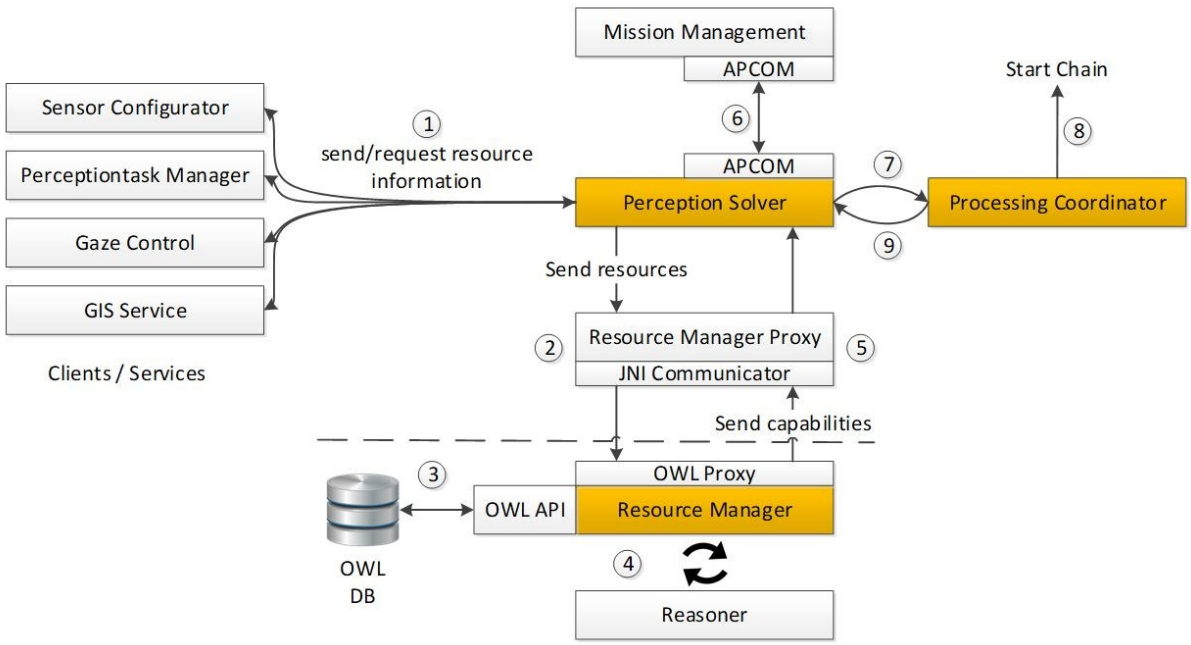


Abbildung 47: Kommunikationsablauf zwischen *Perception Solver*, *Resource Manager* und *Processing Coordinator*

Abbildung 47 zeigt den Ablauf zwischen den drei Kernapplikationen (gelb). Wird der *Perception Solver* initialisiert, melden sich verschiedene *Client*-Applikationen (*Sensor Configurator*, *Perceptiontask Manager, Gaze Control*, *GIS Service*, etc.) am *Solver* an, je nachdem welche Services, basierend auf der Plattformkonfiguration, zur Verfügung stehen. Die *Clients* übermitteln ihre Ressourcen an den *Solver*. Zwischen *Client* und *Solver* besteht eine bidirektionale Kommunikationsverbindung (1); sowohl *Solver* als auch Client können entweder Daten an den jeweils anderen übermitteln oder Informationen anfragen. Ändert sich der Zustand im *Solver*, wird dieser an die Clients übermittelt. Je nachdem welche *Clients* betroffen sind, kann der *Solver* nicht nur die Zustandsänderung übermitteln, sondern auch zusätzliche Informationen bei den *Clients* anfragen. Findet eine Veränderung bei den *Clients* statt, wird der *Solver* darüber informiert. Fällt z.B. ein Sensor aus, ist der *Sensor Configurator* in der Lage, diese Veränderung festzustellen und informiert daraufhin den Solver, welcher Sensor nicht mehr zur Verfügung steht.

Die vom *Solver* empfangenen Ressourcenaufstellungen werden an den *Resource Manager Proxy* (2) übermittelt, woraufhin dieser diese aufbereitet, konvertiert und über eine JNI (*Java Native Interface*) Schnittstelle an den *Resource Manager* übermittelt. Der *Resource Manager* verwaltet die Ontologie und bietet Schnittstellen, um mit der Wissensdatenbank (*OWL DB*) zu interagieren (3). Anschließend werden mittels Inferenzmechanismen (*Reasoner*) die Beziehungen der Individuen untereinander inferiert (4) und der neue Zustand der Ontologie (z.B. vorhandene Fähigkeiten) wieder an den *Resource Manager Proxy* übermittelt (5). Dieser stellt die empfangenen Daten dem *Solver* bereit.

Das *Mission Management* kommuniziert über die *APCOM* Schnittstelle mit dem *Solver*. Auf der einen Seite übermittelt der *Solver* dem MMS Beschreibungen von Fähigkeiten, die zur Verfügung stehen, und auf der anderen Seite delegiert das MMS Perzeptionsaufgaben an den *Solver* (6). Der *Solver* wertet die vom MMS übermittelte Aufgabe aus, entscheidet aufgrund der gespeicherten Fähigkeitenbeschreibung, welche Ressourcen notwendig sind und ermittelt anschließend die zur Erfüllung der Aufgabe geeigneten Perzeptionsketten. Da an dieser Stelle mehrere Perzeptionsketten zur Auswahl stehen können, findet eine Bewertung der Ketten statt (siehe [61]), bevor die zu startende Kette an den *Processing Coordinator* übermittelt wird (7). Dieser prüft, ob die Modulbeschreibungen vollständig sowie die Verknüpfungen der einzelnen Module innerhalb der Kette valide sind und alle Module gestartet werden können. Werden keine Fehler festgestellt, startet der *Coordinator* die Kette (8) und informiert den *Solver* darüber (9).

Sowohl der *Perception Solver* als auch der *Processing Coordinator* besitzen Informationen darüber, welche perzeptive Aufgabe gerade durchgeführt wird und welche Ressourcen die verschiedenen Perzeptionsketten beziehen.

6.2 Ontologieumsetzung und -integration

In der Aufgabenstellung (Kapitel 1.4) wurden bereits zwei Anwendungsfälle für den Einsatz des Ressourcenmodells vorgestellt:

1. *Während der Missionsdurchführung*: Im Sinne des Sensor- & Perzeption Managements soll das Modell an Bord eines spezifisch ausgerüsteten Luftfahrzeuges zur Bestimmung einzelner geeigneter Ressourcenkombinationen herangezogen werden.
2. *Vor der Missionsdurchführung*: Um eine allgemeinere, breitere Verwendbarkeit des Modells zu demonstrieren, soll es bereits vor der Durchführung der eigentlichen Mission angewendet werden. Hierbei sollen unter Berücksichtigung gegebener Aufklärungsaufgaben die Auswahl eines geeigneten UAV, die mitzuführende Sensornutzlast sowie letztlich die für die Aufgabe mitzuführenden Verarbeitungsfunktionen bestimmt werden.

Durch eine holistische Beschreibung von Ressourcen, Fähigkeiten und deren Abhängigkeit untereinander wird sichergestellt, dass die Anforderungen beider Anwendungsfälle erfüllt werden können [45].

Um dies zu gewährleisten, wurde im ersten Schritt eine geeignete *Taxonomie* gewählt. Diese dient der Strukturierung und Sortierung aller Konzepte, wie z.B. Fähigkeiten und Ressourcen, die notwendig sind, um die Beziehung untereinander zu beschreiben. In Abbildung 48 ist die vollständige Klassenhierarchie des Ressourcenmodells (verwendete Klassenkonzepte aus Ontologien Dritter sind der besseren Übersicht halber ausgeblendet) dargestellt. Die Ontologie besitzt über 100 Klassenkonzepte.

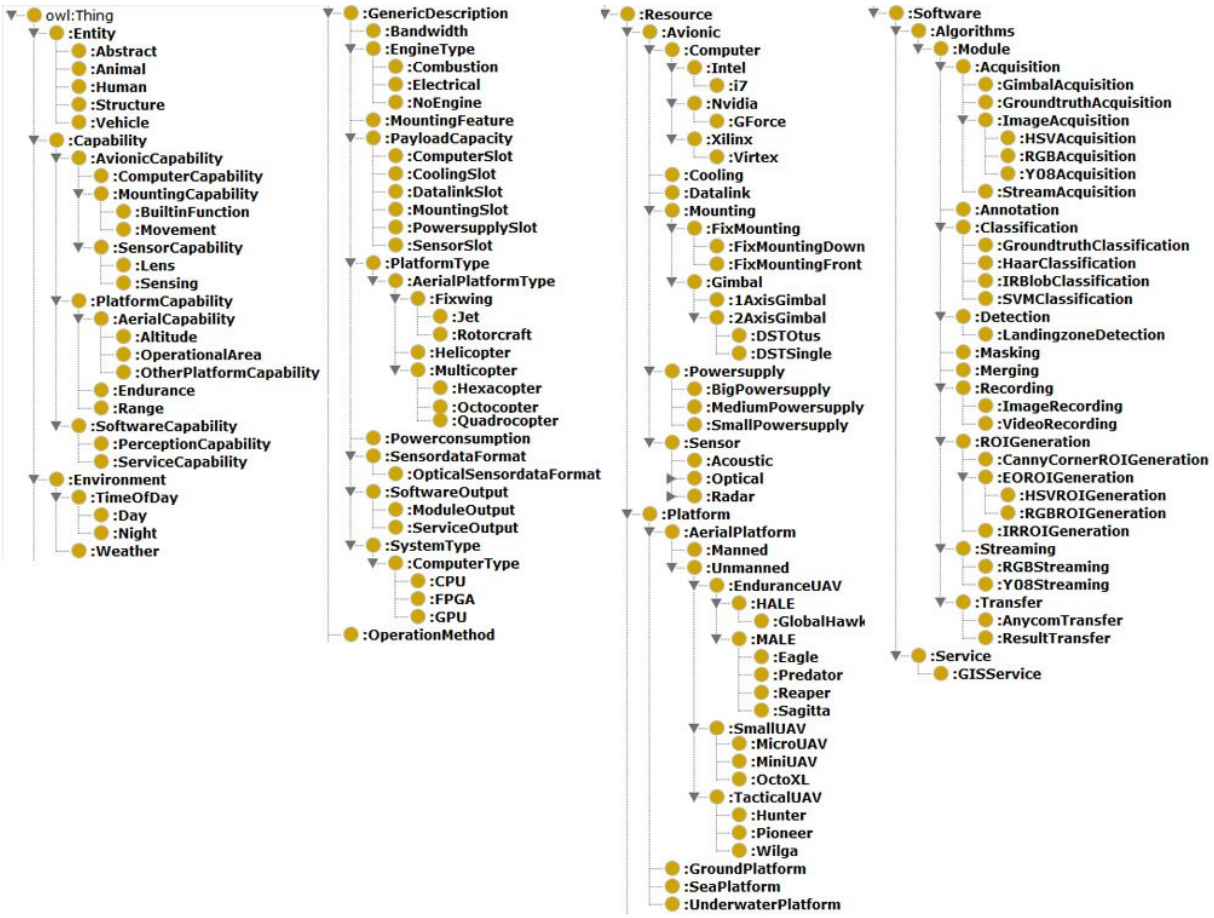


Abbildung 48: Vollständige Klassenhierarchie der Ontologie zur Ressourcenmodellierung

Im Folgenden werden die einzelnen Klassen und deren Bedeutung im Detail betrachtet. In Abbildung 49 sind die Klassen der obersten Hierarchie dargestellt. Von diesen Klassen erben alle weiteren Klassen.

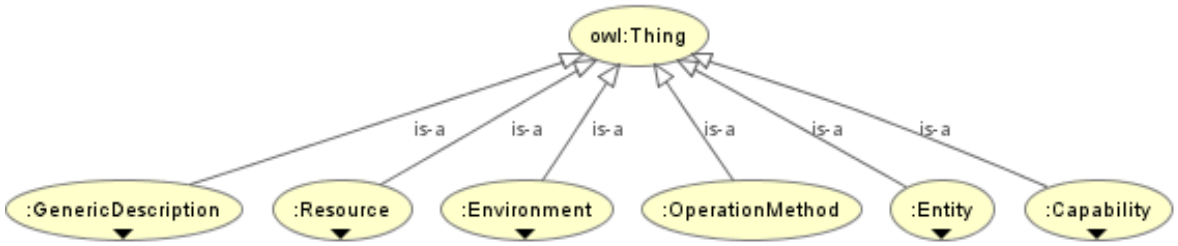


Abbildung 49: Oberste Klassenhierarchie

GenericDescription beinhaltet Klassen, die zur allgemeinen Beschreibung von Sachverhalten oder Eigenschaften benötigt werden. Zu diesen zählen z.B. die Ressourcen In/Outputs, wie sie in Kapitel 4.4 vorgestellt wurden. Weiterhin beinhaltet die Klasse Subklassen zur Beschreibung verschiedener Triebwerkstypen und deren Eigenschaften, Plattformtypen (z.B. *Fixedwing*, *Helicopter* und *Multicopter*), Sensordaten-Formate, Computerhardwaretypen (FPGA, CPU, GPU) und viele weitere.

Die Klasse *Resource* fasst alle Klassen zusammen, die für die Beschreibung von Ressourcen notwendig sind. In Abbildung 50 sind die Ressourcen mit einem Klassenradius von drei dargestellt. Die Ressourcen werden dabei in drei Gruppen unterteilt: *Avionic*, *Software* und *Platform*.

Abbildung 50: Klassenhierarchie der Ressourcen.

Die Klasse der Plattform wird vollständigkeitshalber in vier verschiedene Kategorien unterteilt. Von diesen ist im Rahmen dieser Arbeit jedoch nur die Klasse *AerialPlatform* von Interesse. Diese Klasse dient der Strukturierung verschiedener Luftfahrzeuge und deren Klassifizierung wie z.B. *HALE*, *MALE*, *MicroUAV*, etc. (vgl. Abbildung 51). Die restlichen drei Klassen wurden für mögliche zukünftige Erweiterungszwecke beibehalten.

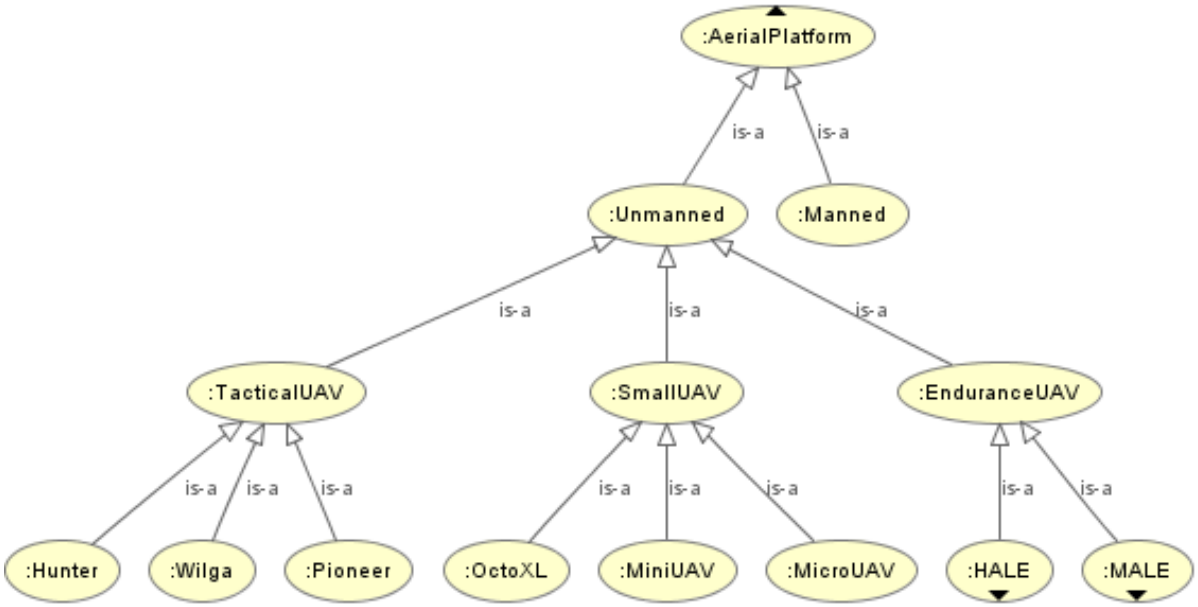


Abbildung 51: Klassenhierarchie für unterschiedliche Luftfahrzeuge und deren Klassifizierung. Zu beachten ist, dass Klassen sowohl allgemeine Subklassen als auch konkrete Klassen von UAV-Typen besitzen können. Zu sehen ist dies in der Abbildung an der Klasse *SmallUAV,* die sowohl allgemeine Unterklassen (*MiniUAV*, *MicroUAV*) als auch Klassen von konkreten UAV-Typen (*OctoXL*) besitzt.

Die Klasse *Environment* dient der Beschreibung von Umgebungscharakteristika wie z.B. Wetter, Tageszeit und Jahreszeit. Weiterhin gehören hierzu Beschreibungen der räumlichen Umgebung (*rural*, *urban*) .

Die Klassen *Entity* und *OperationMethod* dienen der Beschreibung von Fähigkeiten, die von Sensoren bzw. Modulen zur Verfügung gestellt werden. Genauer gesagt, ergibt sich die Fähigkeit einer dieser Ressourcen aus den Instanzen, die innerhalb dieser zwei Klassen modelliert sind. Entitäten repräsentieren Objekte, die von Aufklärungssystemen detektiert, verfolgt und identifiziert werden sollen. Bespiele für solche Objekte sind Personen, Fahrzeuge, Gebäude und Konkretisierungen dieser, wie z.B. Zivilist oder der Fahrzeugtyp Bus. Zu den Methoden (Abbildung 49: *OperationMethod*) gehören unter anderen „*detektieren*“, „*identifizieren*“, „*streamen*“ und „*tracken*“. Die Fähigkeit eines Perzeptionsmoduls stellt sich aus dessen Methode, z.B. „*detektiert*“, und den Entitäten zusammen, z.B. „Zivilist“.

Neben den Fähigkeiten von Perzeptionsmodulen gibt es eine Reihe von Fähigkeiten, die von HW-Ressourcen aufgrund z.B. ihrer Funktionsweise oder ihres Designs bereitgestellt werden. Beispiele dafür sind die Fähigkeit eines Sensors, aufgrund eines Objektivs zu zoomen oder die Fähigkeit eines Multicopters, die Position zu halten („*hovern*“). Diese Art von Fähigkeiten werden in der Klasse *Capability* beschrieben. In Abbildung 52 ist ein Ausschnitt der Klassenhierarchie für die Klasse *Capability* dargestellt.

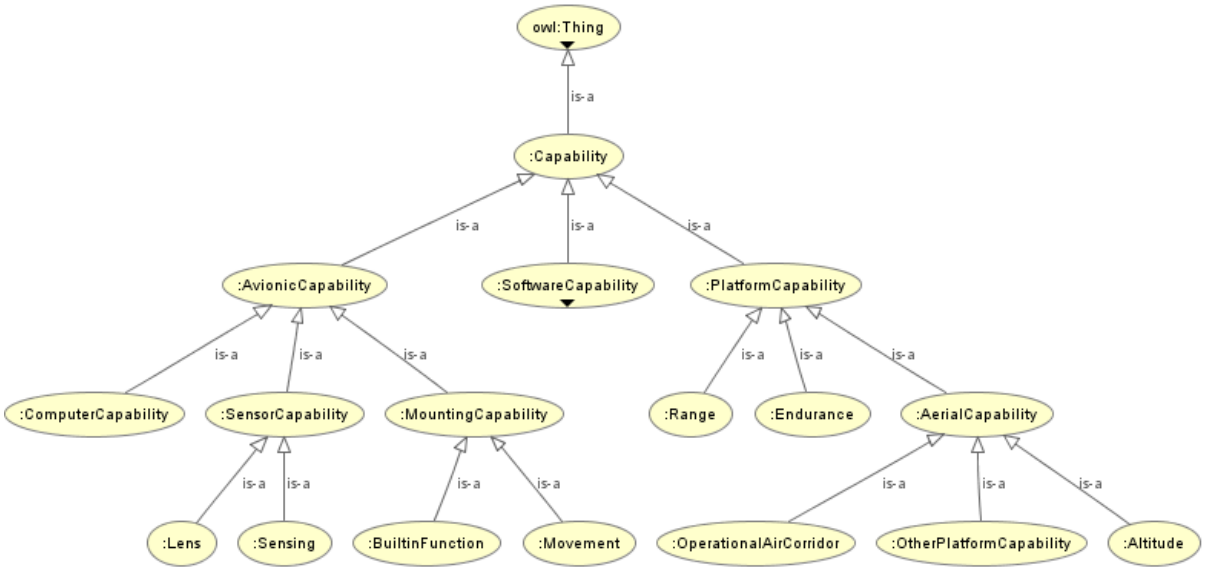


Abbildung 52: Hierarchie der Klasse *Capability*

Erkennbar ist, dass sich diese Fähigkeiten ebenfalls in Subfähigkeiten aufteilen, um so die Fähigkeiten der verschiedenen Ressourcen besser klassifizieren zu können. *AvionicCapability* teilt sich z.B. in *SensorCapability*, *ComputerCapability* und *MountingCapability* und beschreibt damit einen Teil der Ressourcenfähigkeiten, die durch die Integration der Ressourcen an Bord eines Luftfahrzeugs zur Verfügung gestellt werden. *SensorCapability* beinhaltet die Beschreibung der Fähigkeiten von Sensoren, z.B. welche Eigenschaften das Objektiv eines Sensors mitbringt oder *was* ein Sensor wahrnehmen kann (z.B. Wärmesignaturen). In der Klasse *ComputerCapability* werden die Charakteristika verschiedener potentieller Computerarchitekturen (CPU, GPU, FPGA, etc.) beschrieben. *MountingCapability* dient der Fähigkeitenbeschreibung von Sensorhalterungen, wie z.B. Gimbals. Gimbalsysteme unterscheiden sich in ihrer Form, Größe, Ausstattung und Funktionalität. Sensorstabilisierung, Geolock und die Verwendung verschiedener mechanischer Freiheitsgrade sind Beispiele für Fähigkeiten, die von einem Gimbal bereitgestellt werden können.

PlatformCapability wird ebenfalls in Subklassen unterteilt. Während Elemente in den Klassen *Range* und *Endurance* nicht nur luftfahrtspezifischen Plattformen (z.B. können autonome Fahrzeuge, Wasserfahrzeuge und Luftfahrzeuge eine bestimmte Distanz (*Range*) über eine bestimmte Dauer (*Endurance*) zurücklegen) zugeordnet werden können, sind Fähigkeiten der Klasse *AerialCapability* ausschließlich Luftfahrzeugen zuzuordnen. Damit werden die in dieser Klasse beschriebenen Fähigkeiten eingeschränkt. Zum Beispiel hat ein UAV die Fähigkeit, eine bestimmte Flughöhe (*Altitude*) zu halten oder in einem Luftraum zu operieren (*OperationalAirCorirdor*).

6.2.1 Ontologieintegration Dritter

Wie bereits in Kapitel 5.3.3 festgestellt, gibt es eine Reihe von bereits verfügbaren Ontologien, die Konzepte beschreiben, welche für das im Rahmen dieser Arbeit entwickelte Ressourcenmodell von Interesse sind. Dennoch gibt es keine Ontologie, die allen Anforderungen genügt.

Zur Steigerung der Interoperabilität und zur Integration des Ressourcenmodells in ein breiteres Anwendungsspektrum wurden im Rahmen dieser Arbeit auch bereits bestehende Ontologien aufgegriffen und Teile davon in die zu entwickelnde Gesamtontologie integriert. Zur Beschreibung der Aufklärungssensorik wurde auf Konzepte aus der *Semantic Sensor Network* (SSN) [110] Ontologie zurückgegriffen. Abbildung 53 veranschaulicht die Klassen und Beziehungen der SSN Ontologie. Die SSN Ontologie übersteigt die benötigte Beschreibungstiefe von Sensoren, so wie sie im Rahmen dieser Arbeit genutzt werden. Trotzdem ermöglicht die Integration der SSN in die eigene Ontologie und das Verknüpfen des Konzeptes *Sensor* zwischen beiden Ontologien eine detailliertere Beschreibung und demonstriert eine solche Integrationsmöglichkeit für zukünftige Erweiterungen.

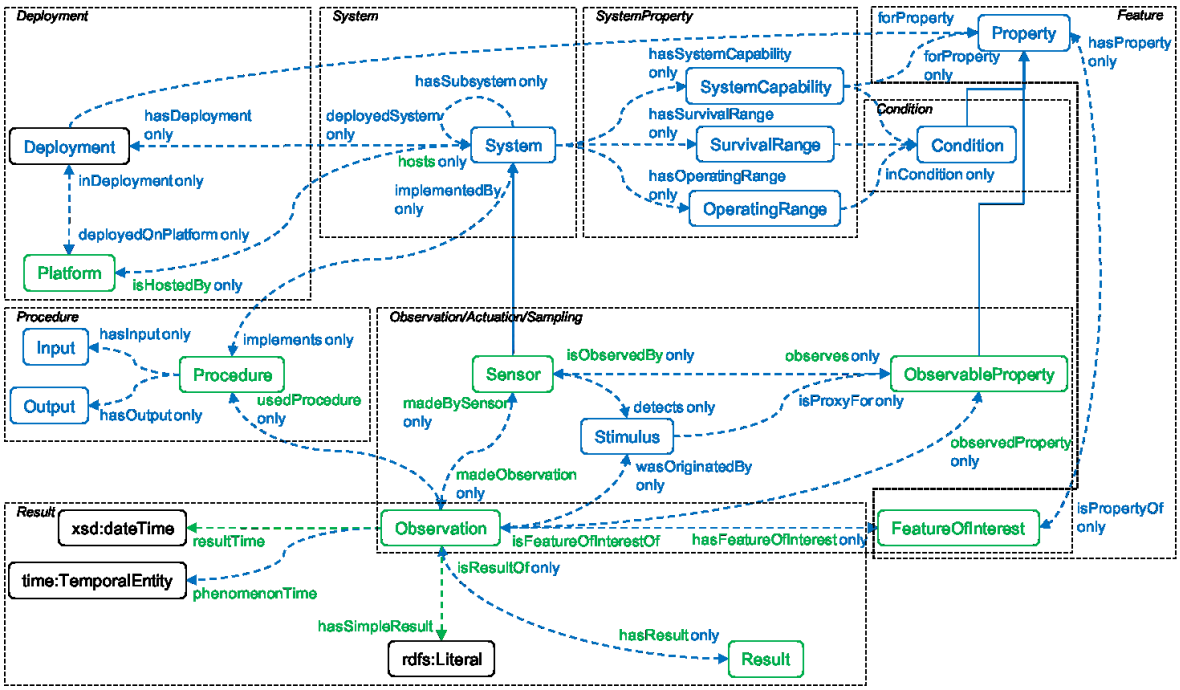


Abbildung 53: Klassen und Rollen der SSN Ontologie [116]

Die SSN Ontologie nutzt Teile der *DUL* (*Dolce-UltraLite*) Top Level Ontologie. Um die Kompatibilität zu wahren und nicht verschiedene Top Level Ontologien miteinander zu vermischen, wurden zur Beschreibung von Top Level Konzepten ebenfalls Teile der DUL Ontologie importiert. Der Zweck der DUL Ontologie kann in einem Satz zusammengefasst werden:

> *"To provide a set of upper level concepts that can be the basis for easier interoperability among many middle and lower level ontologies"* [117]

Abbildung 54 illustriert die Klassen der DUL Ontologie und die Integration der SSN Ontologie Klassen.

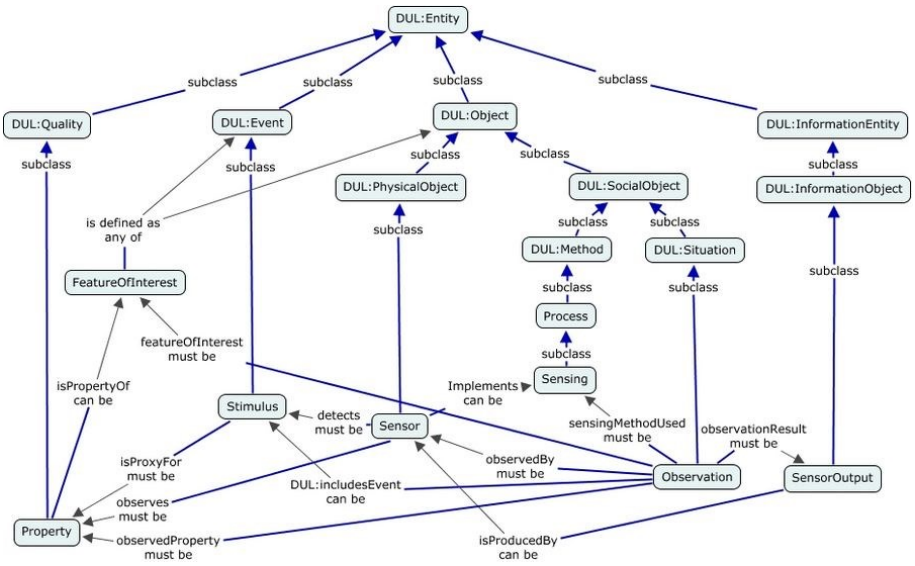


Abbildung 54: Klassen der DUL Ontologie und die Integration der SSN Klassen [118]

Es wird deutlich, dass die Klasse *Sensor* aus der SSN Ontologie eine Subklasse der *DUL:PhysicalObject* Klasse ist. Ein Ausschnitt der Integration der DUL Ontologie in die Ontologie für das Ressourcenmodell ist in Abbildung 55 zu sehen. Die Klasse *Avionic* und *Plattform* sind Subklassen von *DUL:PhysicalObject*.

Abbildung 55: Einordnung von verschiedenen Konzepten zur DUL Ontologie

Neben der SSN und DUL Ontologie wurden zwei weitere Ontologien, die in Eigenentwicklung am Institut entstanden sind, integriert. Bei der Ersteren handelt es sich um eine Ontologie zur Beschreibung von Fuzzykonzepten. Diese werden im Rahmen einer anderen Forschungsarbeit

[119], die auf Erkenntnissen aus dieser Arbeit aufbaut, benötigt, um eine anschließende Bewertung von Perzeptionsketten durchführen zu können. In Abbildung 56 ist das Fuzzy *Software* und *Module* Konzept zusehen, welches in die bestehenden Konzepte über eine *is-a* Beziehung integriert wird.

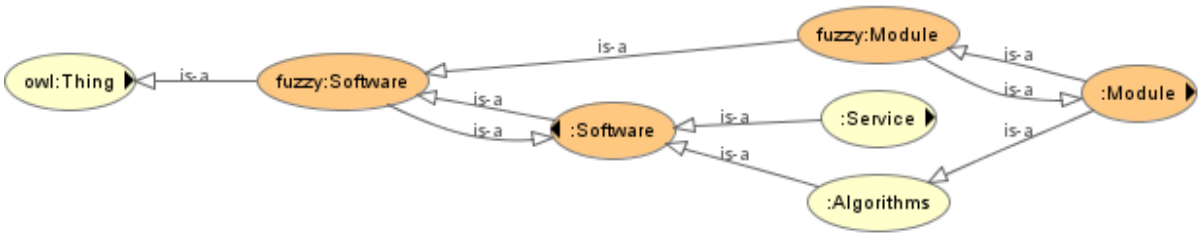


Abbildung 56: Integration der Fuzzy Ontologie in bestehende Konzepte

Abbildung 57 zeigt einen Ausschnitt der Klassenhierarchie des *GenericConcepts* der Fuzzy Ontologie. Diese sind notwendig, um die unterschiedlichen Module in Abhängigkeit von verschiedenen Faktoren zu bewerten.

Abbildung 57: GenericConcept Konzept der Fuzzy Ontologie

Detailliertere Informationen zur Funktionsweise der Perzeptionskettenbewertung und dem Einsatz der Ontologie finden sich in [61], [119] .

Die letzte integrierte Ontologie beschreibt das ATA-Kapitel-Schema (*Air Transport Association*) [56] (vgl. Kapitel 3.5) und zielt darauf ab, die zu entwickelnde Ontologie in einen luftfahrtspezifischen Kontext zu stellen und dient der Demonstration der Integration eines weit verbreitenden Industriestandards. ATA-Kapitel bezwecken eine strukturierte Beschreibung von (zivilen) Luftfahrzeug(sub)systemen und werden von Luftfahrzeugherstellern verwendet. Es existiert auch eine Erweiterung der ATA-Kapitel für militärische Luftfahrzeuge. In den ATA-Kapiteln werden Komponenten und Einrichtungen eines Flugzeugs in Gruppen und Untergruppen unterteilt. Ziel ist es, ein einheitliches Arbeiten von z.B. Wartungspersonal oder Konstrukteuren zu ermöglichen. Auf diese Weise soll das Identifizieren und Überprüfen von Flugzeugkomponenten standardisiert und erleichtert werden. Insgesamt gibt es über 100 Subsysteme, die in fünf Hauptgruppen (*Airframe, Structure, Engines, Miscellaneous, Military*) unterteilt werden können.

Die für diese Arbeit relevanten Ressourcenkonzepte des Aufklärungssystems lassen sich dabei verschiedenen ATA-Kapiteln (24: *Electrical Power*, 23: *Communication*, 21 *Air-Conditioning and Pressurization*, 93: *Survailance*, etc.) zuordnen. Implementiert wurde die ATA-Kapitel Ontologie als Taxonomie, und sie dient der Kategorisierung und Gruppierung. Abbildung 58 veranschaulicht die Überschneidung von Konzepten verschiedener Ontologien und zeigt einen Teil der Klassenhierarchie aus der ATA-Kapitel Ontologie.

Abbildung 58: Klassenhierarchie der ATA-Kapitel Ontologie. Auf Level 1 werden alle Hauptklassen der ATA-Kapitel angezeigt. Zugunsten der Übersicht wurden auf Level 2 und den weiteren Ebenen einige Klassen, die den ATA-Kapiteln angehören, ausgeblendet.

Für das ATA-Kapitel *Military*, welches sich in *Survailance, Weapon_System* und *Missile_Drones_Telemetry* unterteilt, gibt es z.B. eine Überschneidung mit anderen Ontologiekonzepten. So kann das Konzept *Sensor* als Subklasse des *Survailance* Konzeptes betrachtet werden (orangene Ellipsen). Selbiges gilt für das *Airframe_Systems*-Kapitel und *Aircraft_General*. Hier können z.B. der *Datalink*, das *Cooling* und die *Powersupply* verschiedenen ATA-Kapiteln zugeordnet werden. Zur besseren Übersicht wurden in Abbildung 58 einige Klassen der ATA-Kapitel ausgeblendet.

6.2.2 Klassenbeschreibung

In Kapitel 5.1.5 und 5.3.2 wurde erläutert, wie Klassen beschrieben und mit Eigenschaften versehen werden können. An dieser Stelle soll die Klassenbeschreibung demonstriert und ihr Nutzen erläutert werden. In Abbildung 59 ist die Klassenbeschreibung eines Perzeptionsmoduls zu sehen, das auf Basis von gelernten Bildmerkmalen mittels einer *Support Vector Maschine* in der Lage ist, Fahrzeuge in Bilddaten zu detektieren.

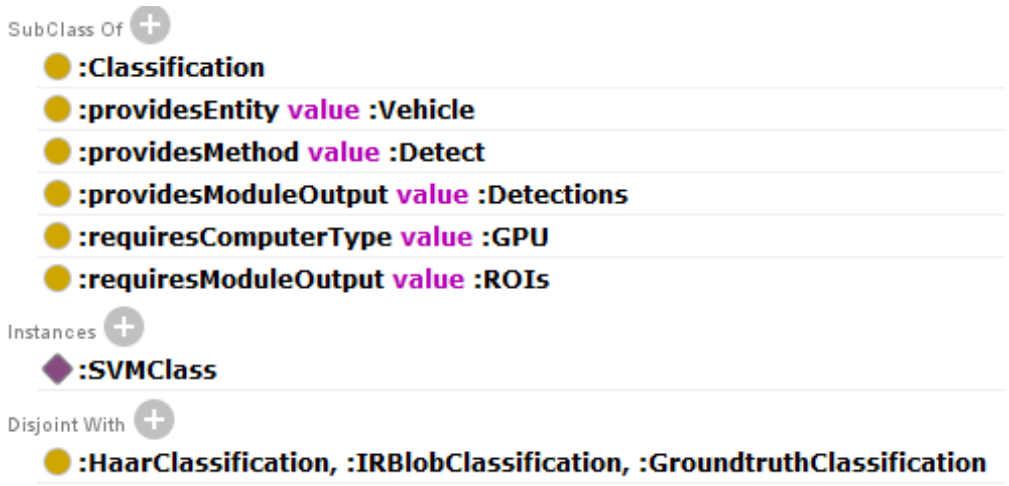


Abbildung 59: Klassenbeschreibung des Perzeptionsmoduls "SVMClassification"

Im unteren Teil der Abbildung (*Disjoint With*) sind disjunkte Klassen gelistet. Sie repräsentieren andere Module, die Objekte mit verschiedenen Methoden detektieren (hier: Haar-Features, IR-Blobs, Groundtruthdaten) können. Aus der Klassenbeschreibung ist abzulesen, dass es eine Instanz dieser Klasse gibt (violette Raute). Unter dem Reiter „SubClass Of“ sind die Klassen und Rollen aufgeführt, die diese Klasse beschreiben. Zu erkennen ist, dass die Klasse eine Kindklasse der Klasse *Classification* ist. Als nächstes wird mit den Rollen *providesEntity* und *providesMethod* beschrieben, dass dieses Modul in der Lage ist, Fahrzeuge (*Vehicle*) zu detektieren (*Detect*). Mittels der Rollen *providesModuleOutput* und *requiresModuleOutput* werden die Daten beschrieben, die das Modul ausgibt bzw. benötigt. Das Ergebnis sind *Detections*. Welches Format diese konkret haben, ist an dieser Stelle nicht ersichtlich. Es wird lediglich festgelegt, dass die Daten der Klasse *Detections* angehören. Als Eingangsdaten benötigt dieses Modul Daten der Klasse *ROIs*. Diese werden von anderen, nicht näher spezifizierten Modulen

zur Verfügung gestellt. Abbildung 60 illustriert als Beispiel die Klassenbeschreibung des Moduls „*RGBROIGeneration*“, welches auf Basis von RGB-Daten ROIs generiert.

SubClass Of
- :EOROIGeneration
- :requiresModuleOutput value :RGBImage

SubClass Of (Anonymous Ancestor)
- :requiresComputerType value :FPGA
- fuzzy:Software
- :providesModuleOutput value :ROIs
- :Software
- :Module
- fuzzy:Module
- fuzzy:hasModuleDescription exactly 1 fuzzy:ModuleDescription

Abbildung 60: Klassenbeschreibung des Moduls "RGBROIGeneration"

Da verschiedene Module zur Ausführung auf unterschiedlicher Hardware implementiert wurden (CPU, GPU oder FPGA), enthält die Klassenbeschreibung die Rolle *requiresComputerType*. Das Modul in Abbildung 59 beispielsweise benötigt eine GPU, während das Modul in Abbildung 60 auf einem FPGA ausgeführt wird.

Abbildung 61 OctoXL: Ein an der Professur für Luftfahrt verwendeter Octocopter

Der Vorteil der Klassenbeschreibung wird besonders dann deutlich, wenn Klassenvererbung eine Rolle spielt. Dies soll an der Klasse *OctoXL*, welche einen Octocopter eines ganz bestimmten Typs repräsentiert, erläutert werden (vgl. Abbildung 61). Die Vererbungsreihenfolge ist wie folgt:

Resource ← Platform ← AerialPlatform ← Unmanned ← SmallUAV ← OctoXL

Abbildung 62 zeigt die relevanten Klassenbeschreibungen der Mutterklassen (linke Seite der Abbildung) und die Beschreibung der Klasse *OctoXL*. Die Klassen auf der linken Seite sind hierarchisch geordnet, d.h. die Beschreibung einer Klasse wird auf die darunterliegende Klasse

vererbt. Zur besseren Übersichtlichkeit wurden die vererbten Rollen bei den Klassen auf der linken Seite nicht dargestellt.

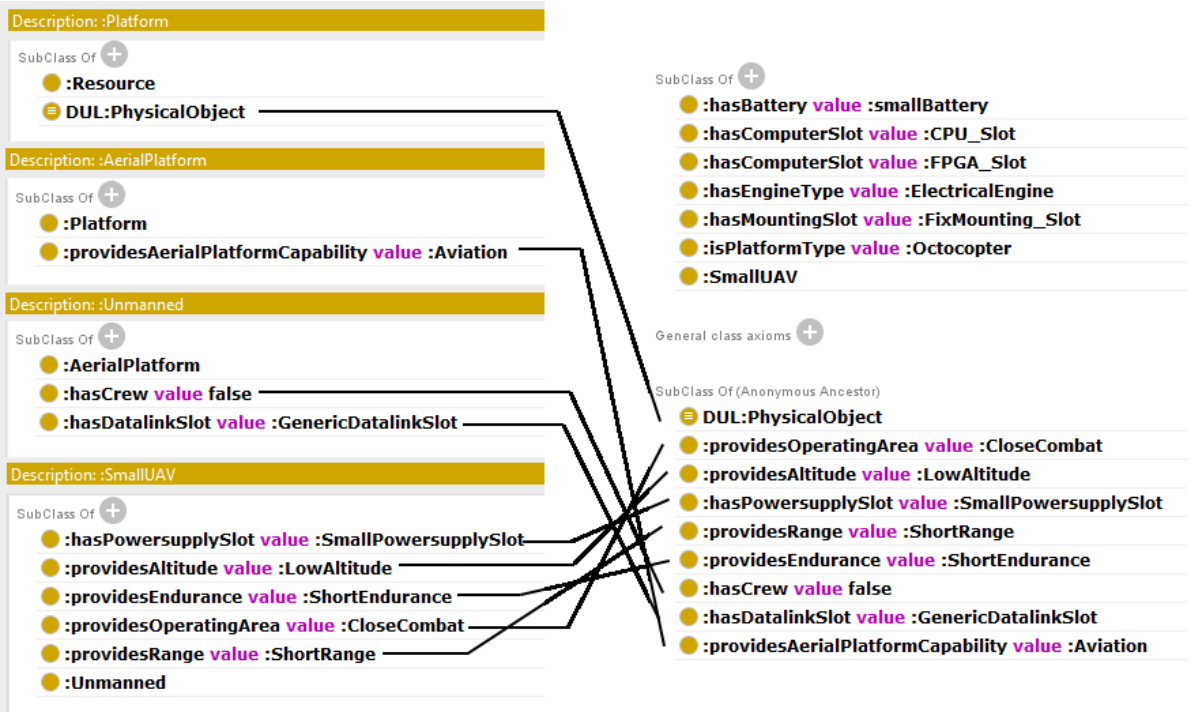


Abbildung 62: Vererbte Rollenbeschreibungen

Für die Klasse *Platform* wird lediglich festgelegt, dass sie eine Subklasse von *Resource* und *DUL:PhysicalObject* ist. Die Klasse AerialPlatform enthält die Rollenzugehörigkeit *providesAerialPlatformCapability*(*Aviation*). Bei der Klasse *Unmanned* gilt, dass für Plattformen dieses Typs keine Besatzung an Bord des Luftfahrzeugs existiert und Platz für einen nicht näher spezifizierten Datenlink vorhanden ist. Bei der Klasse *SmallUAV* beschreiben Rollen ebenfalls unterschiedliche Eigenschaften, wie z.B. die Reichweite (*providesRange*) oder den Operationsbereich (*providesOperatingArea*) dieses UAV-Typs.

Auf der rechten Seite der Abbildung ist die Klassenbeschreibung der Klasse *OctoXL* zu sehen. Im oberen Bereich unter dem Reiter „SubClass Of“ werden verschiedene Eigenschaften aufgelistet, die nur diese Klasse betreffen (z.B. das auf das UAV entweder eine CPU oder ein FPGA montiert werden kann). Die Eigenschaften im unteren Bereich unter dem Reiter „SubClass of (Anonymous Ancestor)“ werden dieser Klasse automatisch durch Vererbung von ihrer Mutterklasse zugewiesen. Würde OWL keine Vererbung unterstützen, müssten diese Eigenschaften für jede Klasse manuell angegeben werden.

Wird nun eine Instanz der Klasse *OctoXL* angelegt, ist der Inferenzmechanismus in der Lage, die Rollen der Klassenbeschreibung dieser Instanz zuzuweisen. Zusätzlich zu der automatischen Zuweisung von Rollen können einer Instanz Rollen auch manuell zugewiesen werden. Dies ist dann sinnvoll, wenn es keine Klasse für einen Sonderfall gibt oder man aufgrund minimaler Abweichungen keine neue Klasse für ein Objekt anlegen möchte. Abbildung 63 zeigt einen Ausschnitt der *OctoXL* Instanz.

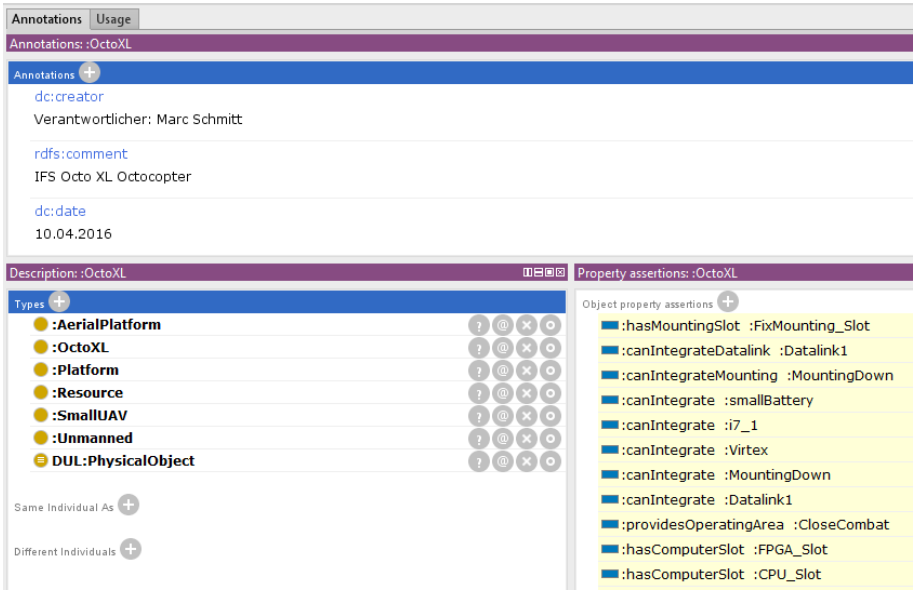


Abbildung 63: Instanz der Klasse *OctoXL*

Der obere Bereich zeigt Annotationen, wie z.B. die verantwortliche Person, einen Kommentar und ein Datum. Links darunter stehen die Klassenzugehörigkeiten. Auf der rechten Seite ist ein Ausschnitt der Rollenzuweisungen abgebildet. Die Liste ist nicht vollständig und zeigt nur einen Ausschnitt. Insgesamt besitzt diese Instanz 36 *Objectproperties* und 5 *Dataproperties*. Zu beachten ist hierbei, dass die Klassenbeschreibung lediglich 13 Rollenbeschreibungen enthält. Aufgrund der in der Klassenbeschreibung angegebenen Rollen ist der Inferenzmechanismus in der Lage, auf weitere Rollen und Eigenschaften zu schließen, z.B. auf Informationen und Zusammenhänge, die aus der reinen Klassenbeschreibung nicht ersichtlich sind und erst zur Laufzeit (aufgrund aller vorhandenen Instanzen) ermittelt werden können.

6.2.3 Data- und Objectproperties

Wie in Kapitel 5.3.2.2 bereits ausgeführt, unterscheidet OWL zwischen Data- und Objectproperties. Dataproperties werden als konkrete und Objectproperties als abstrakte Rollen bezeichnet. Während abstrakte Rollen die Beziehung zwischen zwei Individuen beschreiben, werden konkrete Rollen verwendet, um Individuen mit Datenwerten wie z.B. Zeichenketten, Zahlen, booleschen Ausdrücken oder einer Datumsangabe zu verknüpfen.

xsd:string	xsd:float	xsd:time	xsd:gYear	xsd:gMonth
xsd:anyURI	xsd:NCName	xsd:long	xsd:short	xsd:unsignedLong
xsd:language	xsd:unsignedByte	xsd:boolean	xsd:double	xsd:date
xsd:gMonthDay	xsd:hexBinary	xsd:token	xsd:NMToken	xsd:Name
xsd:int	xsd:byte	xsd:unisgnedInt	xsd:integer	xsd:decimal
xsd:dateTime	xsd:gYearMonth	xsd:gDay	xsd:base64Binary	xsd:normalizedString
xsd:positiveInteger	xsd:nonPositiveInteger	xsd:negativeInteger	xsd:nonNegativeInteger	xsd:unsignedShort

Tabelle 8: Datentypen für konkrete Rollen in OWL [41]

Die Restriktionen bei konkreten Rollen beziehen sich auf den Anwendungsbereich (*Domain Restriction* einer Klasse) und den Geltungsbereich (*Range Restriction* auf Datentypen). Die zur

Verfügung stehenden Datentypen sind in Tabelle 8 dargestellt. Die konkreten Rollen innerhalb der Ontologie werden genutzt, um unterschiedliche nominale Werte von Klassen und Instanzen zu beschreiben. So wird beispielsweise für die Klassen und Subklassen der Plattformen mittels konkreten Rollen festgelegt, welches Gewicht ein UAV hat oder welches Nutzlastgewicht zulässig ist. Im Falle von elektrischer Hardware, wie z.B. einem Gimbal, können die Rollen dazu dienen, neben dem Gewicht der Hardware zu beschreiben, wie viele Sensoren beispielsweise in den Gimbal integriert werden können und wie hoch die Stromaufnahme ist. Diese Daten können später ausgelesen und verarbeitet werden, um verschiedene Aussagen über eine mögliche UAV-Konfiguration zu treffen. Beispielsweise kann so ermittelt werden, ob die Stromversorgung des UAVs für die Nutzlast genug Leistung erbringt oder ob die Nutzlastkonfiguration das zulässige Nutzlastgewicht des UAVs überschreitet. Diese Daten können anschließend in einem grafischen Benutzerinterface visualisiert werden (vgl. Abbildung 64).

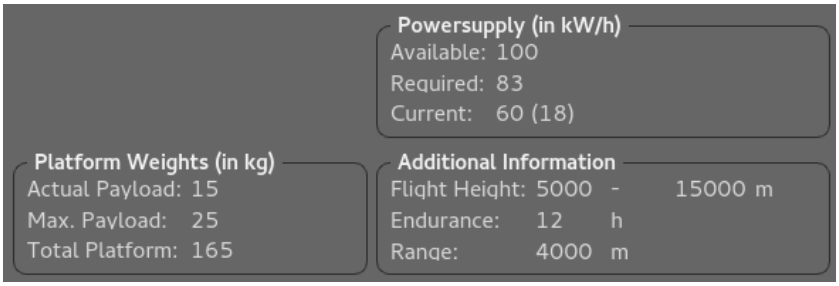


Abbildung 64: Darstellung von akkumulierten Datenwerten aus konkreten Rollenbeziehungen

Abstrakte Rollen beschreiben Beziehungen zwischen Klassen bzw. Individuen. Diese werden verwendet, um mittels des Inferenzmechanismus auf weitere, nicht explizit modellierte Zusammenhänge zwischen Individuen zu schließen. Die Restriktionen (*Domain Restriction* und *Range Restriction*) beziehen sich hierbei in beiden Fällen auf Klassen. Für die Rolle *producesSensordata* wird z.B. als Domain die Klasse *Sensor* und als Range die Klasse *SensordataFormat* festgelegt. Auf diese Weise kann sichergestellt werden, dass diese Rolle nicht an einer falschen Stelle (z.B. in Verwendung mit anderen Klassen) genutzt wird. Wird die Rolle falsch verwendet, kann dies durch den Inferenzmechanismus detektiert und die Ontologie als *inkonsistent* deklariert werden. Weiterhin können den abstrakten Rollen verschiedene Eigenschaften (vgl. Kapitel 5.3.2.2) wie Transitivität, Asymmetrie oder Reflexivität zugeordnet werden. In Abbildung 65 sind die im Rahmen dieser Arbeit notwendigen Rollen dargestellt.

Ebenso wie bei Klassen gibt es auch bei Rollen Hierarchien. An dieser Stelle wird jedoch darauf verzichtet, die Bedeutung jeder Rolle zu erläutern. Stattdessen soll anhand eines Beispiels demonstriert werden, wie die Rollen im Rahmen dieser Ontologie verwendet werden, um nicht explizit modellierte Informationen mittels des Inferenzmechanismus zu generieren. So kann beispielsweise die Beziehung *requiresSlot* näher spezifiziert und in Rollen für verschiedene Slottypen unterteilt werden (*requiresComputerSlot*, *requiresCoolingSlot*, *requiresSensorSlot*, etc.).

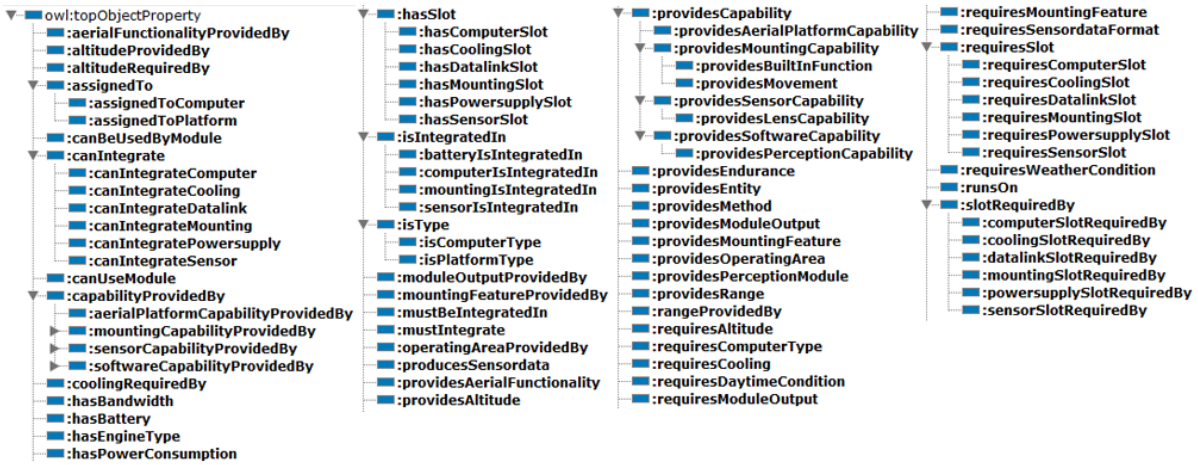


Abbildung 65: Objectproperties

Möchte man eine konkrete Aussage darüber treffen, dass eine Ressource z.B. den Typ *ComputerSlot* benötigt, kann die Beziehung *requiresComputerSlot* verwendet werden. Möchte man umgekehrt nur wissen, welche Slots benötigt werden, kann nach der Beziehung *requiresSlot* gesucht werden. Über die zugeordnete Klasse lässt sich anschließend identifizieren werden, um welchen Slottyp es sich handelt:

```
requiresSlot(Gimbal, Mounting_Slot)
```

In dem oberen Beispiel würde man über die Beziehung *requiresSlot* herauslesen können, dass ein *Gimbal* einen *Mounting*_Slot benötigt, damit dieser in eine Plattform eingebaut werden kann.

Die Klasse *PayloadCapacity* (vgl. Abbildung 48) und deren Kindklassen (*ComputerSlot*, *DatalinkSlot*, etc.) sind Hilfskonstrukte, um auszudrücken, welche Komponenten in eine Ressource verbaut werden können. Dies ist notwendig, da andernfalls für jede Klasse manuell modelliert werden müsste, welche anderen Komponenten zu dieser in Bezug stehen. Mit Hilfe des Slot-Konzeptes übernimmt diese Aufgabe die Inferenzmaschine. So wird z.B. für die Klasse Gimbal mittels der Beziehung

```
hasSensorSlot(Gimbal, Sensor_Slot)
```

festgelegt, dass der Gimbal in der Lage ist, einen Sensor zu integrieren. Um welchen Sensor es sich handelt, ist zu diesem Zeitpunkt nicht bekannt. Der Gimbal selbst besitzt kein Wissen darüber, welche Sensoren vorhanden sind. In der Klassenbeschreibung eines konkreten Kamerasensors (z.B. *SonyFCB*) wird mittels der Beziehung

```
requiresSensorSlot(Sensor, Sensor_Slot)
```

ausgedrückt, dass dieser Sensor in einen Sensorslot verbaut werden kann. Wer diesen Sensorslot zur Verfügung stellt, ist an dieser Stelle nicht bekannt. Auch hier wird lediglich beschrieben, dass der Sensor generell in einem Sensorslot integriert werden kann.

Existiert nun eine Instanz der Klasse *Gimbal* und der Klasse *SonyFCB,* kann mittels der Inferenzmaschine darauf geschlossen werden, dass der Sensor in den Gimbal integriert werden kann. Um dies innerhalb der Ontologie sichtbar zu machen, wird der Instanz der Klasse *Gimbal* bzw. der Klasse *Sensor* eine neue Beziehung

`canIntegrate(Gimbal, SonyFCB)` (und ihre Inverse `isIntegratedIn(SonyFCB, Gimbal)`)

zugeordnet. Die Beziehung *canIntegrate* ergibt sich aus der Verkettung der Beziehungen *hasSensorSlot* und *requiresSensorSlot* bzw. ihrer Inversen. Abbildung 66 veranschaulicht die Funktionsweise.

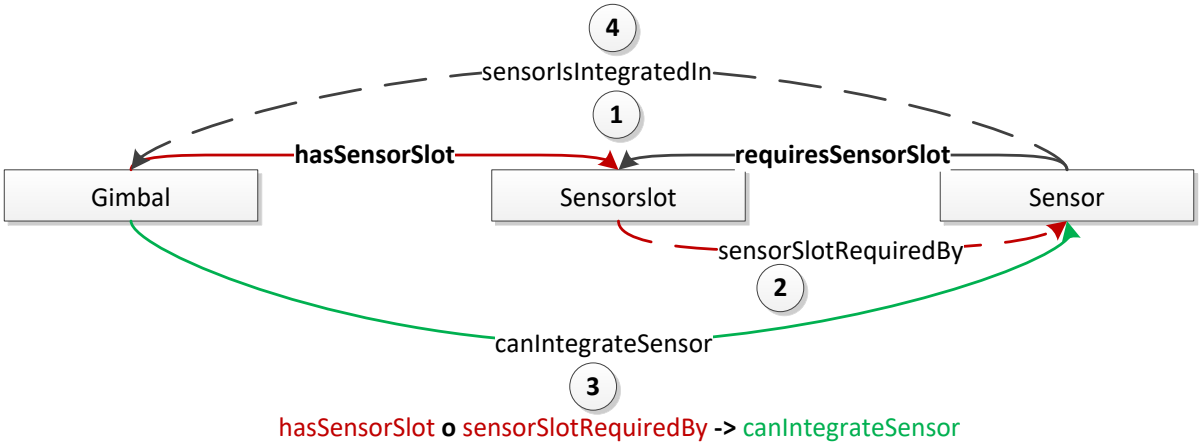


Abbildung 66: Verkettung der Rolle canIntegrateSensor

Die gestrichelten Pfeile repräsentieren die inversen Beziehungen:

- *sensorIsIntegratedIn* <-> *canIntegrateSensor* und
- *requiresSensorSlot* <-> *sensorSlotRequiredBy*.

Aus der Verkettung der mit roten Pfeilen markierten Beziehungen (*hasSensorSlot* und *sensorSlotRequiredBy*) kann auf die grün markierte Beziehung (*canIntegrateSensor*) geschlossen werden. Modelliert sind lediglich die zwei fett markierten Beziehungen (*hasSensorSlot* und *requiresSensorSlot*). Die übrigen Beziehungen werden automatisch aus den zwei angegebenen Beziehungen geschlossen. Die Reihenfolge der Zuweisungen wird durch die Zahlen 1-4 dargestellt: Aus 1 folgt die inverse Beziehung *sensorSlotRequiredBy.* Aus der Beziehung bei 1 und 2 folgt 3 und aus 3 folgt 4.

Die Rollen aus Abbildung 65 können somit in zwei Gruppen unterteilt werden: Rollen, die im Rahmen der Klassenbeschreibung manuell zugeordnet werden und Rollen, die den Klassen und Instanzen durch das Schlussfolgern zugeteilt werden. Das automatische Zuweisen der *canIntegrate* Beziehung ist auch noch einmal in Abbildung 63 dargestellt (in diesem Fall nicht für einen Gimbal, sondern für ein UAV).

6.2.4 Modulkonkatenation

In Kapitel 4.3 wurden die Perzeptionsmodule zur Kapselung von bildverarbeitenden Algorithmen vorgestellt. Aus diesen Modulen lassen sich Perzeptionsketten bilden, die für komplexe Bildverarbeitungsaufgaben zu Aufklärungszwecken genutzt werden. Die Perzeptionsketten selbst sind nicht innerhalb der Ontologie modelliert, sondern ergeben sich zur Laufzeit aus der Verkettung der vorhandenen Ressourcen wie z.B. Sensoren und Perzeptionsmodulen.

Jedes Modul wird in der Ontologie mitsamt seinen Eigenschaften modelliert. Zu den Daten zählen z.B. Parameter zum Steuern des Algorithmus oder die ID, damit bei der Aktivierung einer Kette erkannt werden kann, welche Module gestartet werden sollen. Daneben enthält die Beschreibung eines Moduls dessen Eingangs- und Ausgangsdaten. Diese werden in einer separaten Klasse (*ModuleIO*) instanziiert. Über die Instanzen der Klasse *ModuleIO* kann identifiziert werden, welche Module miteinander interagieren können, bzw. in welcher Reihenfolge die Daten von Modul zu Modul übergeben werden müssen. Das direkte Verknüpfen von Modulen innerhalb der Ontologie ist nicht möglich, da zur Beschreibungszeit nicht bekannt ist, welche Module bzw. Sensoren vorhanden sind bzw. während einer Mission vorhanden sein werden. Werden Module und Sensoren während der Laufzeit der Ontologie hinzugefügt bzw. entfernt, bleiben die Instanzen der Klasse *ModuleIO* immer ein Teil der Ontologie und können somit immer referenziert werden. Abbildung 67 veranschaulicht diesen Unterschied anhand von vier Beispielen.

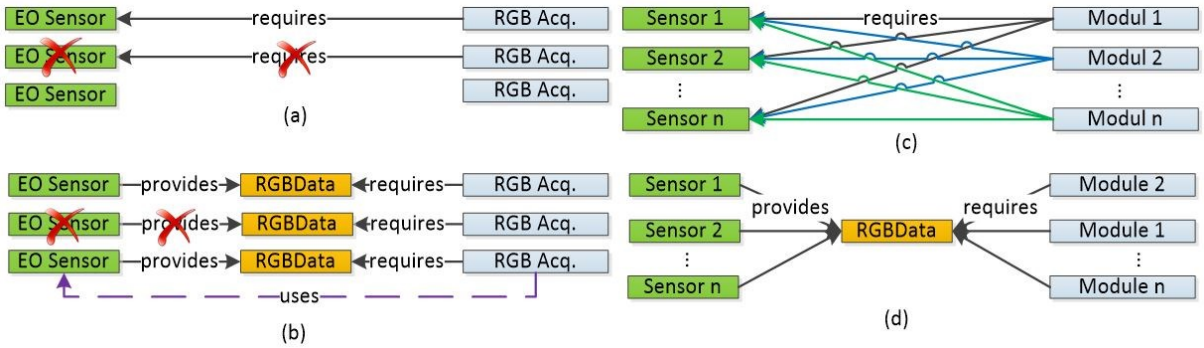


Abbildung 67: (a): Verknüpfung von zwei Modulen auf direktem Weg. (b): Verknüpfung von zwei Modulen über eine Instanz der Klasse *ModuleIO*. (c): Direkte Verknüpfung bei einer unbekannten Anzahl von Sensoren und Modulen. (d): Verknüpfung unbekannter Module und Sensoren über eine Instanz der Klasse *ModuleIO*.

In (a) wird für ein Modul, welches RGB-Daten (*RGB Receiver*) entgegennimmt, direkt mit einem EO-Sensor verknüpft. In OWL würde dies wie folgt beschrieben:

```
1  <owl:NamedIndividual rdf:about="RGB Receiver">
2      <rdf:type rdf:resource="Module"/>
3      <owl1:requires rdf:resource="EO-Sensor"/>
4  </owl:NamedIndividual>
```

Listing 10: Zuweisung eines EO-Sensors zu einem Verarbeitungsmodul

Wenn der EO-Sensor ausfällt und deshalb aus der Ontologie entfernt wird, kann Zeile 3 aus Listing 10 nicht mehr aufgelöst werden, da „EO-Sensor“ in der Ontologie nicht mehr existiert.

Aus diesem Grund wird die Zeile in der Beschreibung des *RGB Receiver* Modules entfernt. Wird der Sensor wieder hinzugefügt, kann das Modul darüber nicht informiert werden, da die Beschreibung der Beziehung innerhalb des Moduls und nicht des Sensors steht. Theoretisch bestünde die Option, die Beschreibung sowohl im Modul als auch im Sensor zu halten. Da es jedoch eine unbekannte Anzahl an Modulen und Sensoren gibt, die miteinander interagieren können (vgl. (c)), würde die Beschreibung zum einen sehr unübersichtlich werden. Zum anderen wäre es nicht möglich, jede Konfiguration zu beschreiben, da Anzahl und Konfigurationsmöglichkeiten der Individuen unbekannt sind bzw. ständig variieren.

```
1   <owl:NamedIndividual rdf:about="RGB Receiver">
2       <rdf:type rdf:resource="Module"/>
3       <owl1:requires rdf:resource="RGBData"/>
4       <owl1:uses rdf:resource="EO Sensor"/>
5   </owl:NamedIndividual>

6   <owl:NamedIndividual rdf:about="EO Sensor">
7       <rdf:type rdf:resource="Sensor"/>
8       <owl1:provides rdf:resource="RGBData"/>
9       <owl1:usedBy rdf:resource="RGB Receiver”/>
10  </owl:NamedIndividual>
```

Listing 11: Zuweisung eines EO-Sensors zu einem Verarbeitungsmodul über ein persistentes Individuum

In (b) erfolgt die Verknüpfung von Modulen und Sensoren über eine Instanz der *ModuleIO* Klasse. In diesem Beispiel symbolisiert die Instanz ein RGB-Daten Format. Die Beschreibung der Beziehung aus (b) ist in Listing 11 zu sehen. Wird der Sensor in diesem Fall entfernt, ändert sich nichts an der Modulbeschreibung. Wird der Sensor wieder hinzugefügt, wird auch die Beschreibung aus Zeile 9 wieder hinzugefügt. Mittels Inferenz (vgl. Kapitel 6.2.3) kann anschließend auf die `uses` Beziehung geschlossen werden (Zeilen 4 und 10 in Listing 11 und die lila Verbindung in Abbildung 67 (b)). Weiterhin spielt es nun auch keine Rolle, wie viele Sensoren bzw. Module vorhanden sind. Die Ressourcen beschreiben lediglich, welche *ModuleIO* Daten sie bereitstellen bzw. benötigen (vgl. (d)). Die restlichen Beziehungen (z.B. die `uses`-Beziehung) werden von der Inferenzmaschine ergänzt. Abbildung 68 veranschaulicht diese Vorgehensweise.

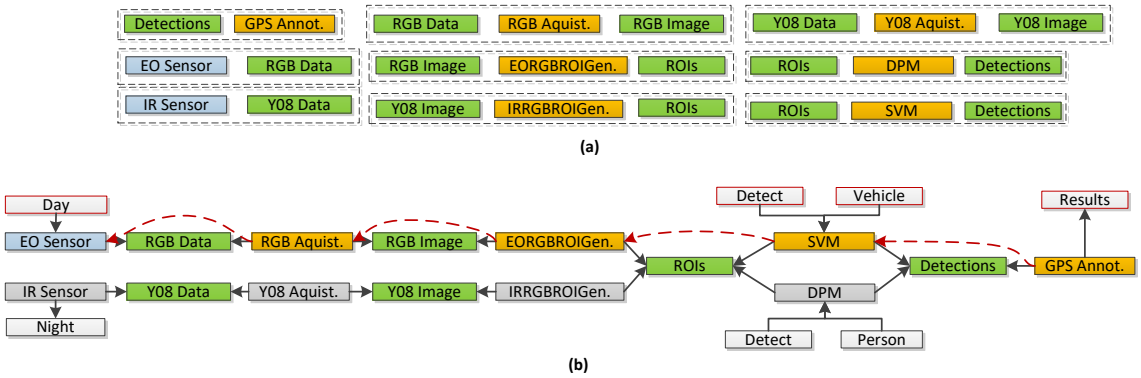


Abbildung 68: Verknüpfung von Modulen zu einer Perzeptionskette

In der Ontologie liegt nun die Beschreibung der Module und Sensoren mit ihren Eingangs- und Ausgangsdaten (a). Soll eine Perzeptionsaufgabe durchgeführt werden (Bsp.: „*Fahrzeug bei Tag detektieren und die Ergebnisse speichern“),* werden zunächst die Ressourcen identifiziert, die die notwendigen Fähigkeiten anbieten (im angegebenen Beispiel die Module *SVM*, *GPS Annotation* und der Sensor *EO-Sensor*). Ausgehend von diesen Informationen wird, beginnend mit dem letzten Modul, aufgrund der *ModuleIO* Beschreibung identifiziert, welche weiteren Module benötigt werden (b). Über den Inferenzmechanismus werden zusätzliche Rollen zugewiesen, die die Module direkt miteinander verbinden. In dem Beispiel ergeben sich aufgrund der *ModuleIO* Beschreibungen verschiedene Traversierungen. Jedoch kommen einige Module (grau) wegen der Beschreibung der Perzeptionsaufgabe nicht in Frage. In Abbildung 69 sind die Rollenzuweisungen aus dem Beispiel in Abbildung 68 für das *SVM* Modul beschrieben.

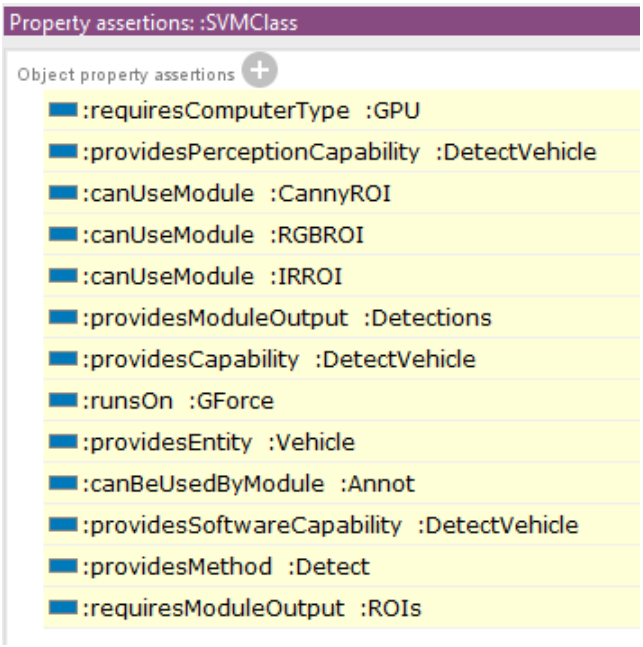


Abbildung 69: Inferrierte Rollen des Moduls *SVMClass*

Aus der Klassenbeschreibung des Moduls folgen nun die Rollenzuweisungen *requiresModuleOutput* und *providesModuleOutput*. Aus diesen wird anschließend geschlossen, dass dieses Modul mit den Modulen (*canUseModule*) *CannyROI*, *RGBROI* und *IRROI* verknüpft werden kann. Das Modul selbst kann von dem *Annotation* Modul (*canBeUsedByModule*) genutzt werden. Die Fähigkeiten (*providesCapability*) des Moduls werden durch die Rollen *providesEntity* und *providesMethod* bereitgestellt.

6.2.5 Ontologieinstanzen

Die Instanzen können in zwei Kategorien unterteilt werden: Instanzen, die permanent Bestandteil der Ontologie sind (z.B. Fähigkeiten) und Instanzen, die abhängig von der Konfiguration bzw. genutzter Software und Hardware dynamisch der Ontologie hinzugefügt bzw. aus dieser wieder entfernt werden (z.B. Ressourcen).

Im Folgenden sollen die Instanzen der zu entwickelnden Ontologie sortiert nach Klassen gelistet werden. Da die Instanzen nicht auf der Ebene der Mutterklasse angelegt werden, wird im Folgenden zu jeder Mutterklasse eine Abbildung der Klassenhierarchie gezeigt, bevor auf die einzelnen Instanzen eingegangen wird. Die Instanzen werden nach folgenden Mutterklassen sortiert (vgl. Abbildung 70): *Environment, Capability, GenericDescription, OperationMethod and Entity, Resource – Aviation, Resource - Platform, Resource – Software*.

Environment

Abbildung 71 zeigt die Klassenhierarchie für die Klasse Environment.

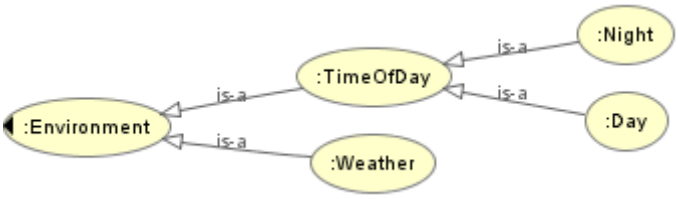


Abbildung 71: Klassenhierarchie: Environment

Die Instanzen dieser Klassen werden genutzt, um Beschränkungen für Ressourcen zu beschreiben. Die Klassen besitzen folgende Instanzen: *Day, Night, Cloudy, Rainy, Sunny*.

Capability

In Abbildung 72 ist die Klassenhierarchie für die Klasse *Capability* dargestellt. Instanzen dieser Klasse werden genutzt, um die Fähigkeit einer Ressource zu beschreiben.

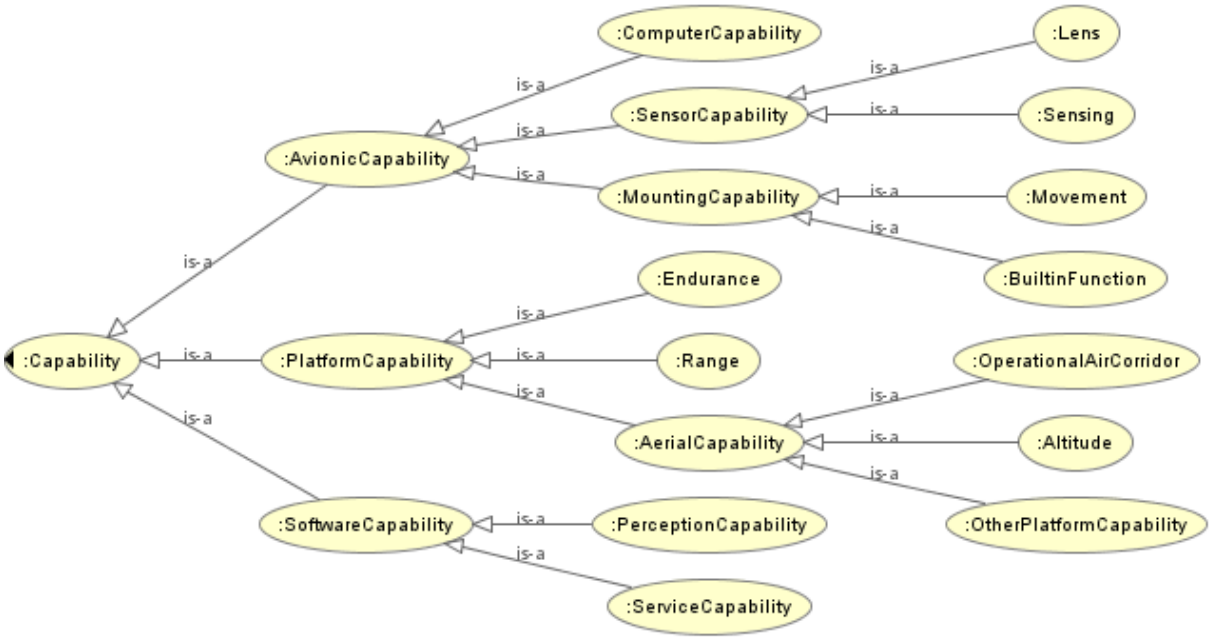


Abbildung 72: Klassenhierarchie: Capability

Tabelle 9 zeigt die Instanzen der Klasse Capability. Insgesamt besitzt die Klasse 44 Instanzen.

Name	Klasse	Name	Klasse
SensorMovement	MountingCapability	MediumEndurance	Endurance
Geolock	BuiltinFunction	ShortEndurance	Endurance
SensorStabilization	BuiltinFunction	LongRange	Range
Tracking	BuiltinFunction	MediumRange	Range
MoveHorizontal	Movement	ShortRange	Range
MoveVertical	Movement	DetectPerson	PerceptionCapability
VisibleImage	SensorCapability	DetectVehicle	PerceptionCapability
ThermalImage	SensorCapability	IdentifyBus	PerceptionCapability
Zooming	Lens	IdentifyCar	PerceptionCapability
SenseInfrared	Sensing	IdentifyCivilian	PerceptionCapability
SenseRadar	Sensing	IdentifySoldier	PerceptionCapability
SenseVisibleLight	Sensing	IdentifyTank	PerceptionCapability
Aviation	PlatformCapability	RecordImage	PerceptionCapability
HighAltitude	Altitude	RecordVideo	PerceptionCapability
LowAltitude	Altitude	RGBStreaming	PerceptionCapability
MediumAltitude	Altitude	StoreResults	PerceptionCapability
AboveClouds	OperationalAirCorridor	StreamResults	PerceptionCapability
BeneathClouds	OperationalAirCorridor	TrackPerson	PerceptionCapability
CloseCombat	OperationalAirCorridor	TrackPOI	PerceptionCapability
HoldPosition	OtherPlatformCapability	TrackVehicle	PerceptionCapability
VTOL	OtherPlatformCapability	Y08Streaming	PerceptionCapability
LongEndurance	Endurance	QueryGISDatabase	ServiceCapability

Tabelle 9: Instanzen der Klasse Capability

OperationMethod and Entity

Die Instanzen der Klasse *OperationMethod* und *Entity* werden genutzt, um zu beschreiben, was vereinzelte Module leisten können. *OperationMethod* beschreibt, welche Methode ein Modul zur Verfügung stellt. Dies kann eine der folgenden sein: *detect, track, identify, record, stream*. Die Klasse *Entity* hingegen beschreibt, was mittels der Methode wahrgenommen wird. Instanzen dieser Klasse sind: *image, result, video, animal, dog, civilian, human, soldier, airport, building, church, bus, car, tank, vehicle*. Die Instanzen der beiden Klassen werden zu einem Tupel der folgenden Form kombiniert (vgl. Kapitel 4.4 für mehr Details):

$$Fähigkeit = \left\{ \begin{matrix} Methode \\ Entität|Daten \\ \sum Beschränkungen \end{matrix} \right\}$$

Die Klasse *Resource* wird im Folgenden übersichtshalber in die drei Hauptkategorien *Avionic, Platform* und *Software* unterteilt. Die Klassenzuordnungen aus Ontologien von Dritten (siehe Kapitel 6.2.1) wurden in den Abbildungen ausgeblendet.

GenericDescription

Abbildung 73 stellt die Klassenhierarchie für die Klasse *GenericDescription* dar. Die Instanzen dieser Klasse sind notwendig, um allgemeine Strukturen und Abhängigkeiten zu beschreiben.

Abbildung 73: Klassenhierarchie: GenericDescription

In Tabelle 10 sind die insgesamt 52 Instanzen der Klasse *GenericDescription* gelistet.

Name	Klasse
HighBandwidth	Bandwidth
LowBandwidth	Bandwidth
Middlebandwidth	Bandwidth
CombustionEngine	Combustion
ElectricalEngine	Electrical
NoEngine	NoEngine
MoveSensor	MountingFeature
Orientation_Back	MountingFeature
Orientation_Down	MountingFeature
Orientation_Front	MountingFeature
Orientation_Left	MountingFeature
Orientation_Right	MountingFeature
Orientation_Up	MountingFeature
CPU_Slot	ComputerSlot
FPGA_Slot	ComputerSlot
GPU_Slot	ComputerSlot
GenericCooling_Slot	CoolingSlot
GenericDatalink_Slot	DatalinkSlot
FixMounting_Slot	MountingSlot
GenericGimbal_Slot	MountingSlot
OtusGimbal_Slot	MountingSlot

Name	Klasse
Jet	Jet
Rotorcraft	Rotorcraft
Helicopter	Helicopter
Hexacopter	Hexacopter
Octocopter	Octocopter
Quadrocopter	Quadrocopter
HighPwrConsumption	Powerconsumption
LowPwrConsumption	Powerconsumption
MediumPwrConsumption	Powerconsumption
BayerBGData	OpticalSensordataFormat
HSVData	OpticalSensordataFormat
RGBData	OpticalSensordataFormat
Y08Data	OpticalSensordataFormat
AnnotatedData	ModuleOutput
Detections	ModuleOutput
GimbalData	ModuleOutput
GroundtruthData	ModuleOutput
HSVImage	ModuleOutput
ImageMask	ModuleOutput
RGBImage	ModuleOutput
ROIs	ModuleOutput

BigPowerSupply_Slot	PowersupplySlot	Y08Image	ModuleOutput
GenericPowersupply_Slot	PowersupplySlot	GISData	ServiceOutput
MediumPowersupply_Slot	PowersupplySlot	CPU	CPU
SmallPowersupply_Slot	PowersupplySlot	GPU	GPU
Sensor_Slot	SensorSlot	FPGA	FPGA

Tabelle 10: Instanzen der Klasse GenericDescription

Vereinzelte Instanzen besitzen eine detailliertere Beschreibung. Diese beinhaltet unter anderem Verweise auf weitere Instanzen (z.B. Fähigkeiten) und numerische Daten. Abbildung 74 einhaltet die Beschreibung für die Instanz *Hexacopter* (vgl. Tabelle 10).

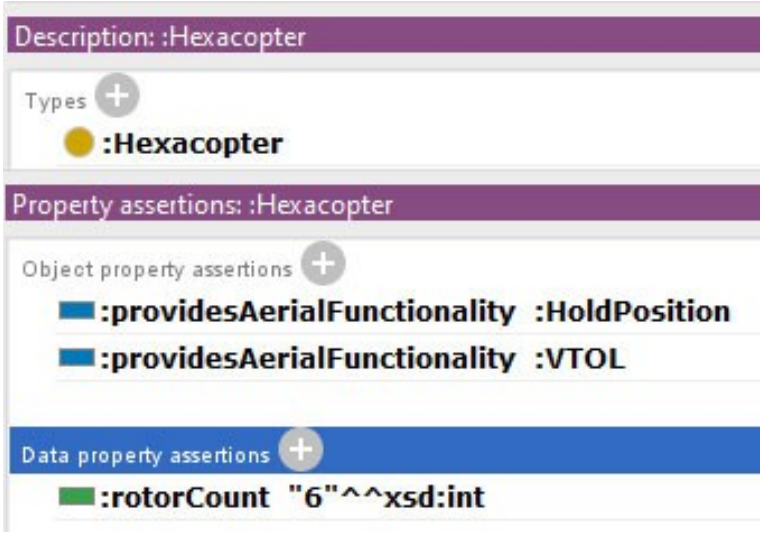


Abbildung 74: Ontologiebeschreibung für die Instanz Hexacopter

Der Beschreibung ist zu entnehmen, dass diese Instanz der Klasse *Hexacopter* angehört (vgl. Abbildung 73). Weiterhin ist zu sehen, dass diese Instanz zwei Beziehungen zu weiteren Instanzen besitzt (vgl. Tabelle 9 – *HoldPosition* und *VTOL*). Darüber hinaus besagt die Beschreibung, dass die Anzahl der Rotoren 6 ist. Wird nun festgelegt, dass eine Plattform vom Typ *Hexacopter* ist, wird automatisch inferriert, dass diese Plattform die genannten Eigenschaften aus Abbildung 74 besitzt.

Resource – Avionic

In Tabelle 11 sind die Instanzen der Klasse Avionic dargestellt.

Name	Klasse	Name	Klasse
i7_1	i7	BigBattery	BigPowersupply
Gforce_1	Gforce	MediumBattery	MediumPowersupply
Virtex_1	Virtex	SmallBattery	SmallPowersupply
GenericCooling	Cooling	SonyFCB6300	SonyFCB
Datalink1	Datalink	ProsilicaGen	Prosilica
FixMountingDown	FixMountingDown	FlirTau	FLIR
DSTOtus1	DSTOtus	SarSensor	Sar
DSTSingle1	DSTSingle		

Tabelle 11: Instanzen der Klasse Avionic

Auch hier sind die Instanzen wieder mit spezifischen Beziehungen und numerischen Daten beschrieben, sodass detaillierte Informationen z.B. beim Abfragen der Datenbank extrahiert werden können. Für EO und IR-Sensoren werden im Rahmen dieser Arbeit die Informationen aus Tabelle 12 in der Ontologie abgelegt.

Name	Typ	Bildwiederholrate	Auflösung	Gewicht	Feature	Wahrnehmung	Power
SonyFCB 6300	EO	30 Hz	1920 x 1080	270 g	zooming	Tageslicht	6.0 - 12.0 V DC
Flir Tau 640	IR	30 Hz	640 x 480	55 g		Temperatur	4.4 - 6.0 V DC

Tabelle 12: Technische Daten von EO und IR-Sensoren

In Abbildung 75 ist die Klassenhierarchie der Klasse Avionic dargestellt.

Abbildung 75: Klassenhierarchie: Avionic

Resource – Platform

Abbildung 76 illustriert die Klassenhierarchie für die Klasse *Platform*.

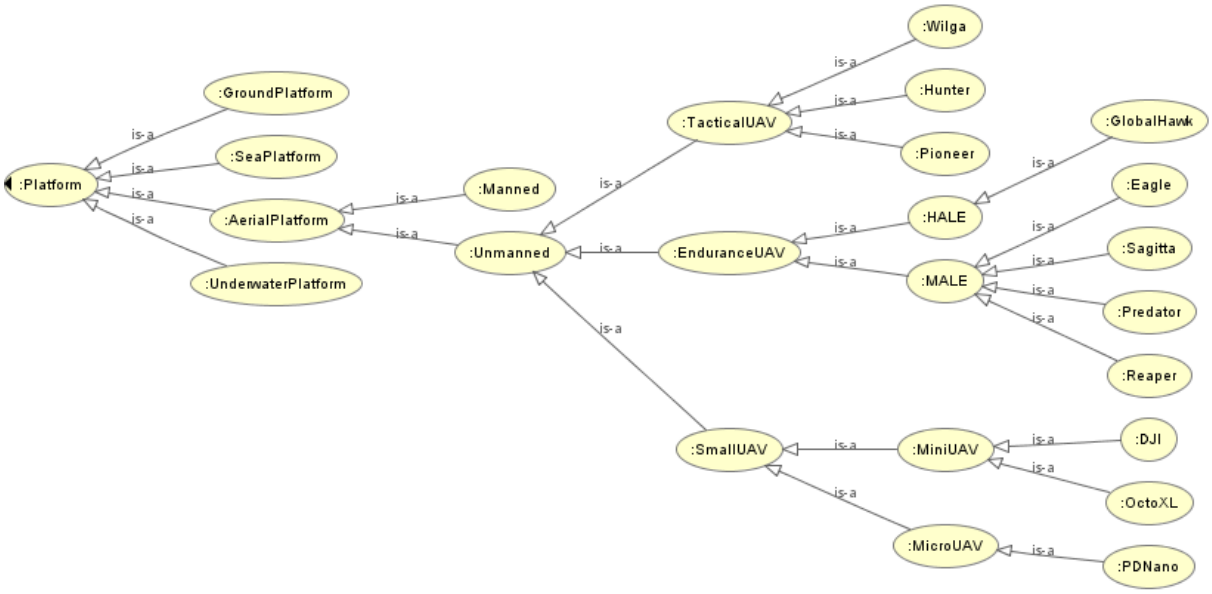


Abbildung 76: Klassenhierarchie: Platform

Insgesamt werden 11 verschiedene UAVs modelliert. Die Eigenschaften der UAVs sind in Tabelle 13 und Tabelle 14 aufgeführt.

Name	GlobalHawk-GER1	Sagitta-1	Eagle-E1	Predator-P1	Reaper-X2
Klasse	HALE	MALE	MALE	MALE	MALE
Gewicht (kg)	6700	150	1600	1000	2200
Computerausstattung	CPU/GPU/FPGA	CPU, GPU, FPGA	CPU/GPU	CPU/GPU	GPU, FPGA
Mounting	Gimbal	Gimbal	Gimbal	Gimbal	Gimbal
Payloadgewicht (kg)	6000	25	300	500	1700
Geschwindigkeit (kmh)	600	80	300	165	300
Reichweite (m)	23000	4000	9000	10000	2000
Flughöhe (m)	18000	300	9000	7600	15000
Sensoren	EO/SAR/IR	EO/IR	SAR/IR/EO	SAR/IR	EO/IR/SAR
Eigenschaften					
Einsatzdauer (h)	32+	0,5	25	22	14
Stromversorgung	Rolls-Royce Turbofan	Battery/Jet	Thielert Centurion	Turbochared	Turoprop
Fuel Capacity (L)	4000	50	1600	1000	1400

Tabelle 13: Eigenschaften der modellierten UAVs, erster Teil

Name	Hornet-Nano	OctoXL-M1	DJI-Phantom	Wilga-E1	Hunter-H1	Pioneer-P1
Klasse	Micro	Mini	Mini	Tactical	Tactical	Tactical
Gewicht (kg)	0,2	25	5	50	700	205
Computerausstattung	N/A	1x GPU	1x CPU	2x CPU	CPU	GPU
Mounting	Fix	Fix	Gimbal	Gimbal	Fix	Gimbal
Payloadgewicht (kg)	0,01	3	2	15	90	34

Geschwindigkeit (kmh)	5	25	30	35	167	200
Reichweite (m)	50	200	150	500	1250	1850
Flughöhe (m)	20	200	150	300	5500	4600
Sensoren	EO/IR	EO/IR	EO	EO	EO	EO/EO
Eigenschaften	hover	hover	hover			
Einsatzdauer (h)	0,2	0,15	0,15	0,3	21	5
Stromversorgung	Battery	Battery	battery	Battery	2x Diesel	Piston Engine
Fuel Capacity (L)	N/A	N/A	N/A	N/A	300	50

Tabelle 14: Eigenschaften der modellierten UAVs, zweiter Teil

In Abbildung 77 sind einige UAVs abgebildet. Von links nach rechts, von oben nach unten: Global Hawk, Sagitta, MQ-9 Reaper, Black Hornet, Octocopter, Hunter

Abbildung 77: Verschiedene UAVs[28]

Für die Instanz *OcotXL-M1* wird z.B. festgelegt, dass diese Instanz vom Typ *Octocopter* ist (vgl. Tabelle 10). Dadurch werden dieser Instanz zuzüglich der Eigenschaften aus Tabelle 14 die Eigenschaften aus der Beschreibung der Instanz *Octocopter* zugewiesen. In Abbildung 78 ist der zusammengesetzte Graph für die Instanz OctoXL-M1 dargestellt. Damit der Graph überschaubar bleibt, wurden in der Abbildung nur die relevanten Beziehungen dargestellt.

Die grünen Rechtecke repräsentieren *Dataproperties* zur Beschreibung numerischer Werte, wie z.B. das Gewicht des UAVs, die Anzahl der Rotoren oder die minimale und maximale Flugzeit. In Orange ist die zu beschreibende Instanz dargestellt und in Blau die übergeordnete Klasse, von der diese Instanz erbt. In weiß sind die Abhängigkeiten zu anderen Instanzen aufgeführt. Dadurch, dass *OcotXL-M1* von der Klasse *MiniUAV* erbt, werden die Beschreibungen der Klasse auf die Instanz übertragen. So wird z.B. für die Instanz *OctoXL-M1* festgelegt, wie hoch, wie weit und wie lange das UAV fliegen kann (vgl. Tabelle 14). Aus der Beziehung

```
OctoXL-M1 -> Typ -> Octocopter
```

kann geschlossen werden, dass diese Instanz die Eigenschaften *HoldPosition* und *VTOL* besitzt.

[28] Bildquellen: https://tinyurl.com/yaomaq8v, https://tinyurl.com/yccc7s6m, https://tinyurl.com/ydd62ncu, https://tinyurl.com/y7aqxsbh, 06.05.2018

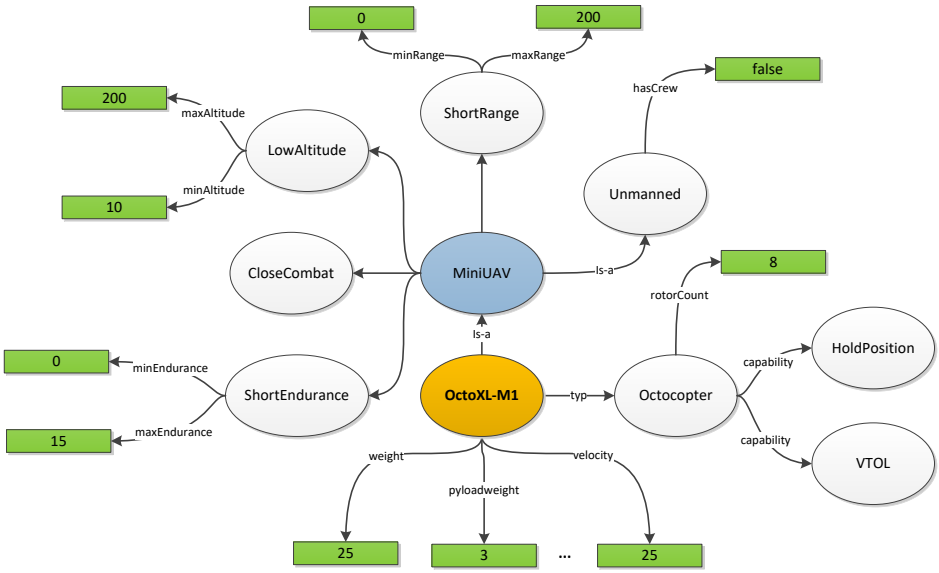


Abbildung 78: Graph für die Beschreibung der Instanz OctoXL-M1

Resource – Software

Abbildung 79 demonstriert die Klassenhierarchie für die Klasse Software. Diese setzt sich aus den Algorithmen respektive Modulen und den Services, wie z.B. dem Geoinformationsservice, zusammen.

Abbildung 79: Klassenhierarchie: Software

In Tabelle 15 sind die Instanzen der Klasse Software aufgelistet. Neben dem Namen und der Klassenzugehörigkeit werden die Fähigkeiten der Module, der Datenoutput und die Vorbedingungen bzw. Anforderungen dargestellt.

Module Name	Klasse	Fähigkeit	Output	Anforderung	Hardware
HSV Acquisition	HSVAcquisition	-	HSVImage	BayerBGData, RGBData	CPU
RGB Acquisition	RGBAcquisition	-	RGBImage	BayerBGData, RGBData	CPU
Y08 Aquisition	Y08Acquisition	-	Y08Image	BayerBGData, RGBData, Y08Data	CPU
SAR Acquisition	SAR Acquisition	-	Pointcloud	SARData	CPU
Annotation	Annotation	Annotate Data	AnnotatedData	Detections	CPU
Haar Classifier	HaarClassification	Vehicle, detect, Identify, Car, Bus	Detections	ROIs	FPGA
IR Blob Detector	IRBlobClassification	Vehicle, Human,Detect	Detections	Y08Image	CPU
DPM Classifier	SVMClassification	Human, Detect, Vehicle	Detections	ROIs	GPU
BG Mask	Masking	-	ImageMask	RGBImage	CPU
Green Mask	Masking	-	ImageMask	RGBImage	GPU
Street Mask	Masking	-	ImageMask	RGBImage	GPU
Image Rec	ImageRecording	Record, Image	-	BayerBGData, RGBData, Y08Data	CPU
Video Rec	VideoRecording	Record, Video	-	BayerBGData, RGBData, Y08Data	CPU
Canny Corner	CannyC.ROI Generation	-	ROIs	ImageMask, Y08Image	GPU
HSV ROI	HSVROI Generation	-	ROIs	HSVImage	FPGA
RGB ROI	RGBROI Generation	-	ROIs	RGBImage	CPU
IR ROI	IRROI Generation	-	ROIs	Y08Image	CPU
RGB Streaming	Streaming	Stream, Video	-	RGBData	CPU
Y08 Streaming	Streaming	Stream, Video	-	Y08Data	CPU
To Anycom	Transfer	Stream, Results	-	AnnotatedData	CPU
Result Transmitter	Transfer	Store, Results	-	AnnotatedData	CPU

Tabelle 15: Instanzen der Klasse Software

Nicht jedes Modul besitzt eine Fähigkeit, da manche Module nur der reinen Datentransformation bzw. Vorverarbeitung dienen. Ebenso produziert nicht jedes Modul einen Datenoutput, der von einem weiteren Modul benötigt bzw. verarbeitet wird. Module, die Bilder, Videos oder Ergebnisse an eine Bodenkontrollstation übertragen oder Daten aufzeichnen, stellen keine Daten zur Weiterverarbeitung bereit, sondern übertragen das Endprodukt einer Perzeptionskette (vgl. Kapitel 4.3 und Kapitel 6.2.4).

Tabelle 16 bildet die Gesamtanzahl aller Axiome der Ontologie ab. Insgesamt besteht die Ontologie aus 3.886 Axiomen, davon beschreiben

- 8.559 Axiome Klassen (bei 414 Klassen),
- 199 Axiome Instanzen (bei 166 Instanzen),
- 656 Axiome Objektbeziehungen (bei 290 Objektbeziehungen),
- 115 Axiome Datenbeziehungen (bei 96 Datenbeziehungen) und
- 1.076 Axiome Annotationen.

Metrics		Object propery Axioms	
Axiom	**3.886**	SubObjectPropertyOf	**188**
Logical axiom count	**1.819**	InverseObjectProperties	**108**
Declaration axioms count	**981**	FunctionalObjectProperty	**1**
Class count	**414**	InverseFunctionalObjectProperty	**1**
Object property count	**290**	ObjectPropertyDomain	**162**
Data property count	**96**	ObjectPropertyRange	**177**
Indiviudla count	**166**	SubpropertyChainOf	**19**
DL expressivity	**SROIQ(D)**		

Class axioms		Data property axioms	
SubClassOf	**727**	SubDataPropertyOf	**12**
EquivalentClasses	**19**	EquivalentDataProperties	**2**
DisjointClasses	**87**	DataPropertyDomain	**10**
Hidden GCI Count	**22**	DataPropertyRange	**91**

Individual axioms		Annotation axioms	
ClassAssertion	**166**	AnnotationAssertion	**1.076**
DataPropertyAssertion	**33**		

Tabelle 16: Verteilung der Axiome innerhalb der Ontologie

Die Ontologie setzt sich wie folgt zusammen:

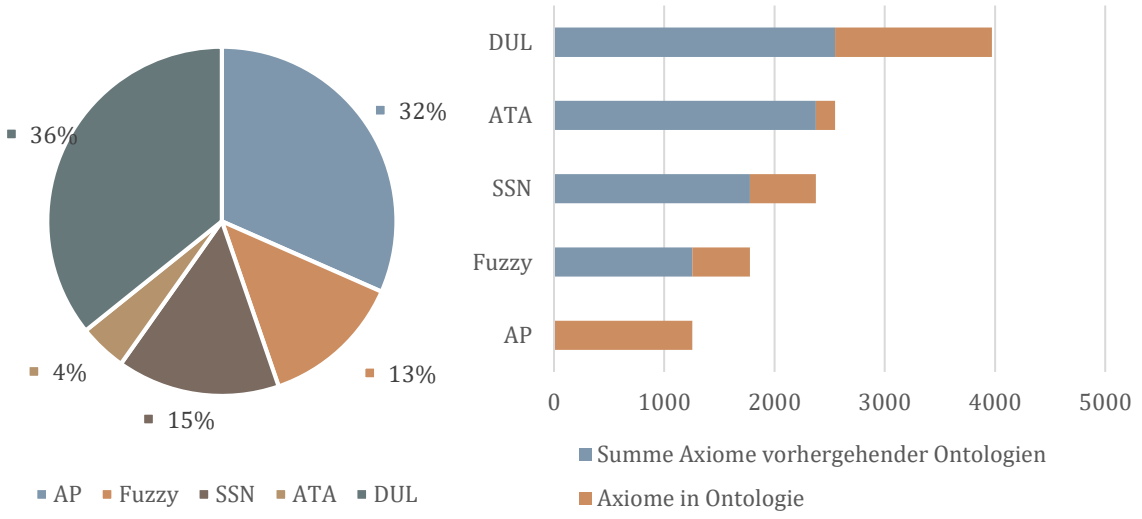


Abbildung 80: Visualisierung der Komplexitätssteigerung durch Hinzufügen externer Ontologien.

Links: der Prozentsatz aller Axiome, verteilt auf die unterschiedlichen Ontologien. Rechts: die Anzahl der Axiome innerhalb einer Ontologie und die daraus resultierende Summe durch Fusionieren der Ontologien.

6.3 Ontologiedatenbank

Eine Ontologie kann auf verschiedene Arten gespeichert werden. Am einfachsten ist es, sie als Datei in einem gängigen Format (Turtle, RDF/XML, OWL/XML, Manchester, etc.) auf einem Laufwerk abzulegen. Enthält die Ontologie eine große Anzahl an Tripel oder soll ein verteiltes Arbeiten an derselben Ontologie ermöglicht werden, bietet es sich an, sie in einer Datenbank zu speichern. Für Ontologien gibt es spezielle Tripel-Speicher Datenbanken (engl. Tripelstore), die im Vergleich zu relationalen Datenbanken performanter sein können, abhängig von der Größe der Ontologie, der Beziehungstiefe der Axiome und Komplexität der Abfragen. In [120] und [121] wird untersucht, inwiefern sich die Performance zwischen einer relationalen Datenbank und einem Tripelstore für Ontologien aus dem Gesundheitswesen unterscheidet. In [122] untersuchen C. Martinez-Cruz *et al.* den Unterschied zwischen Ontologien und relationalen Datenbanken, um Wissen zu speichern. In [123] findet ein Vergleich zwischen RDF und NoSQL[29] Datenbanken statt. In [124] wird untersucht, wie gut NoSQL Datenbanken geeignet sind, RDF Daten zu speichern.

Zunächst soll der Unterschied zwischen einer relationalen Datenbank und einem Tripelstore erläutert werden. Der Hauptunterschied besteht darin, wie Daten in der Datenbank abgelegt werden. In relationalen Datenbanken werden sie in Form von Tabellen abgespeichert und an-

[29] NoSQL („*not only SQL*“) Datenbanken sind strukturierte Datenspeicher, die nicht ausschließlich einen relationalen Ansatz verfolgen. D.h. es gibt keine festgelegten Tabellenschemata.

schließend über einen Index (*Primary Key*) adressiert. Möchte man mehrere Zeilen oder mehrere Tabellen miteinander in Beziehung setzen, geschieht dies über (teuer) JOIN Operationen. Bei Tripelstores hingegen werden die Beziehungen zwischen Daten in Form eines Tripels direkt in der Datenbank gespeichert. Das heißt, die Beziehung zwischen Daten kann direkt aus der Datenbank extrahiert und muss nicht erst mit Hilfe von speziellen Operationen identifiziert werden. Das nachfolgende, stark vereinfachte Beispiel soll dies verdeutlichen.

Luftfahrzeuge		
ID_Luftfahzeug	Luftfahrzeug	bemannt
1	Eurofighter E1	True
2	Tornado T2	True
3	Globalhawk G1	False

Piloten	
ID_Pilot	Name
1	Denis S.
2	Christian H.

Zugehörigkeit		
ID	ID_Pilot	ID_Luftfahzeug
1	1	1
2	2	2
3	NULL	3

Tabelle 17: Beispiel für Tabellen in einer relationalen Datenbank

In Tabelle 17 sind drei Tabellen, wie sie in einer relationalen Datenbank vorhanden sein können, dargestellt. Die erste dient der Beschreibung der vorhandenen Luftfahrzeuge, in der zweiten sind die Piloten gelistet und die dritte Tabelle enthält Informationen darüber, welcher Pilot welchem Flugzeug zugeordnet ist. Um herauszufinden, welcher Pilot welches Flugzeug fliegt, müssen die drei Tabellen zusammengeführt werden, damit man über die IDs zum Namen der Luftfahrzeuge und Piloten gelangt. Die Darstellung in einer graphenbasierten Datenbank hingegen ist intuitiver (vgl. Abbildung 81). Zwei Instanzen sind über eine Beziehung miteinander verbunden. Da der Globalhawk keinen Piloten besitzt, fällt diese Beziehung weg.

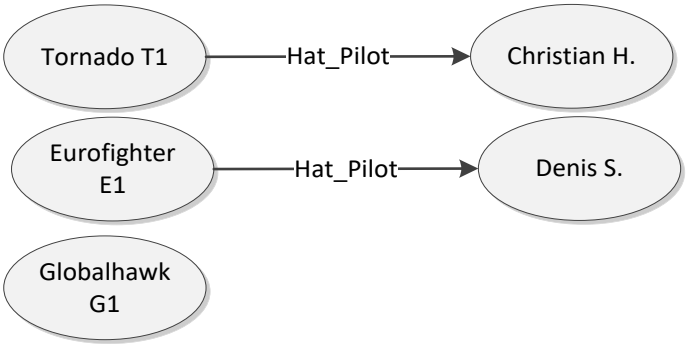


Abbildung 81: graphische Darstellung von Beziehungen in einem Tripelstore

Eine Abfrage für das Beispiel mit der relationalen Datenbank würde aus einer SELECT, FROM, mehreren JOIN und einer WHERE Abfrage bestehen. Für das Beispiel aus dem Tripelstore genügen die folgenden Zeilen:

```
1  SELECT ?p,?o,?v {
2      WHERE ?p Hat_Pilot ?v.
3  }
```

Listing 12: Beispielhafte SPARQL Abfrage

Die Abfragesprache für relationale Datenbanken ist SQL, die für Tripelstores SPARQL.

Relationale Datenbanken sind effizient, solange die abzuspeichernden Daten wenige Beziehungen und Abhängigkeiten untereinander haben. Besonders bei sehr großen Datensätzen bieten relationale Datenbanken einen Geschwindigkeitsvorteil. Sind die Daten jedoch komplex und haben untereinander viele Abhängigkeiten, zeigen sich die Stärken eines Tripelstore, da in diesem Fall keine teuren JOIN Operationen notwendig sind, um Beziehungen zwischen Daten aufzuzeigen.

Die Performance unterscheidet sich jedoch nicht nur zwischen relationalen Datenbanken und Tripelstores, sondern auch von Tripelstore zu Tripelstore. An dieser Stelle sollen nicht weiter die Unterschiede der einzelnen Tripelstores erläutert werden, sondern lediglich, warum die Entscheidung auf den Stardog [125] Tripelstore gefallen ist. Tabelle 18 zeigt einen Auszug aus einem Datenbankvergleich von verschiedenen Tripelstores, der im Jahre 2016 durchgeführt wurde. Verglichen wurden unter anderem der Datenbanktyp, die verwendete Technologiesprache, in der die Datenbank implementiert wurde. Weitere Vergleiche können in [127] eingesehen werden.

	Neo4J	MarkLogic	OrientDB	Stardog
Beliebtheit	34.91	11.20	5.44	**0.41**
Graph DBMS	Ja	Nein	Ja	**Ja**
RDF Store	Nein	Nein	Nein	**Ja**
Tinkerpop	Ja	Nein	Ja	**Ja**
Technologie	Open source Native Graph DBMS	NoSQL DBMS	Multi-Model DBMS	**Semantic Graph DBMS RDF Store**
ACID	Ja	Ja	Ja	Ja
Language	Java, Scala	C++	Java	**Java**
Engine Typ	native	Multi-Model	Multi-Model	**Native**
Bloor Research Zusammenfassung	clear leader for property graph in operational and hybrid operational/analytic	Leader for operational environments and as a unification platform.	Highest ranking in operational and hybrid operational/analytic	**highest ranked product for unification**

Tabelle 18: Vergleich von Tripelstores

In [128] wird die Performance von unterschiedlichen Tripelstores untersucht, indem unterschiedliche SPARQL Abfragen auf den verschiedenen Datenbanken analysiert werden. In [129] wird ein Benchmark für vier Tripelstores mit dem Ergebnis durchgeführt, dass die richtige Wahl eines Tripelstores oft von der Anwendung bzw. dem Fokus abhängt.

Im Ranking[30] der verschiedenen Datenbanken belegt Stardog Platz 6 mit einer Bewertungszahl von 0.58 (Stand September 2017). Laut Stardog-Webseite verwenden einige große Unternehmen die Datenbank, um ihre Ontologien zu speichern. Wie die meisten Tripelstores unterstützt

[30] Siehe: https://db-engines.com/en/ranking/rdf+store

Stardog neben verschiedenen Modellen das RDF Graphen Model, bietet eine SPARQL Schnittstelle und implementiert OWL2. Der Inferenzmechanismus basiert (zu großen Teilen, vgl. Abbildung 82) auf einer Umstrukturierung der Abfrage (*OWL 2 Direct Semantics Entailment Regime* [130]):

> *"Reasoning in Stardog is based (mostly) on a* ***query rewriting*** *technique: Stardog rewrites the user's query with respect to any schema or rules, and then executes the resulting expanded query (EQ) against the data in the normal way. This process is completely automated and requires no intervention from the user."* [125]

Die Schemata und Regeln, nach denen die Abfragen umgeschrieben werden sollen, können vom Benutzer angepasst werden. Es steht ihm zudem frei, die Inferenzmechanismen des Tripelstores zu nutzen oder nicht. Wird *Reasoning* genutzt, geschieht dies im Hintergrund, sodass sich der Benutzer nicht selbst darum kümmern muss, wann und welche Regeln angewendet werden. Dies erlaubt es mit geringfügigem Aufwand, eigene Regeln für Inferenzen zu erstellen. Beispiele und eine strukturierte Dokumentation erleichtern zudem den Einstieg mit dem Stardog Tripelstore.

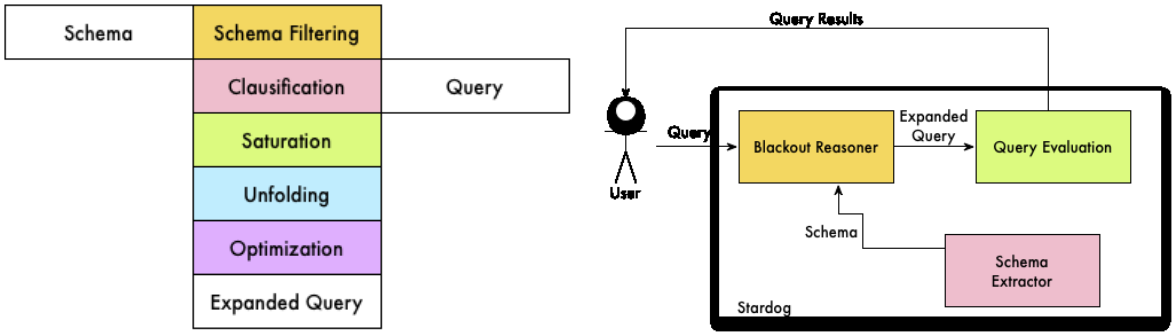


Abbildung 82: Umstrukturierung der Abfrage und das Beantworten der Abfrage in dem Stardog Tripelstore [125]

Ein Webinterface ermöglicht es, die Parameter und das Verhalten der Datenbank zu verändern. Ein SPARQL Editor erlaubt es außerdem, Abfragen „on the fly" an die Datenbank zu senden, wodurch eine schnelles Prototyping und Testen der Ontologie begünstigt wird.

6.4 Umsetzung des Perception Solvers

In Kapitel 6.1 wurde der *Perception Solver* als zentrales Element des S&PM vorgestellt. Der *Solver* empfängt Informationen von allen vorhandenen Ressourcen (z.B. Sensoren, Perzeptionsmodulen, Gimbals, etc.). Er ist außerdem für die Synchronisation und die Verteilung der Daten zuständig. Damit Applikationen (z.B. MMS) außerhalb des S&PM mit dem *Solver* kommunizieren können, fungiert er ebenfalls als APCOM Server (*Active Perception Communication Interface*). Abbildung 83 zeigt die Architektur des *Perception Solvers*.

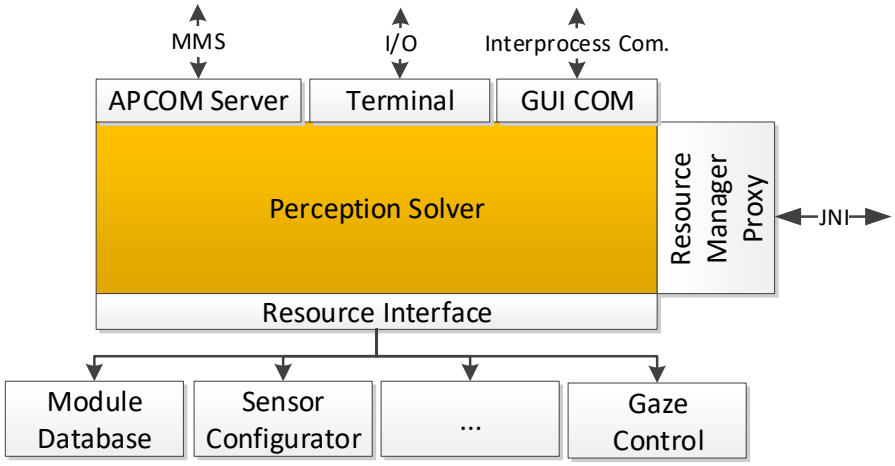


Abbildung 83: Komponenten des Perception Solvers

Der *Solver* verfügt über weitere Schnittstellen, die zu seiner Kommunikation und Steuerung verwendet werden. Die Terminal- bzw. Interprozesskommunikationsschnittstelle ermöglichen es, den *Solver* direkt anzusprechen und so z.B. manuell Ketten zu starten bzw. zu stoppen, den Status des *Solvers* abzufragen (z.B. vorhandene Ressourcen, laufende Ketten, Ontologiezustand) und ihn von einer GUI aus zu steuern bzw. seinen Status zu visualisieren.

Das *Resource Interface* bietet eine gemeinsame Schnittstelle für Applikationen (Clients), die ihre Ressourcen dem Solver bekannt machen möchten und diese innerhalb der Ontologie ablegen wollen. Durch das Interface wird sichergestellt, dass die Daten der verschiedenen Applikationen ein einheitliches Format besitzen. Damit die Kommunikation mit dem *Solver* gewährleistet ist, sieht das Interface die Methoden aus Listing 13 vor. Auf die Beschreibung der einzelnen Methoden wird an dieser Stelle verzichtet, da die Namen selbsterklärend sind.

```
1  bool getAliveStatus();
2  void requestAvailableRessources();
3  void requestRunningRessources();
4  void addRessource();
5  void addRessources();
```

Listing 13: Funktionen, die laut Interface von jedem Benutzer implementiert werden müssen

Aus Kompatibilitätsgründen, auf die im folgenden Kapitel näher eingegangen wird, kann der *Solver* nicht direkt mit der Ontologiedatenbank kommunizieren. Deshalb ist er nicht in der Lage, Informationen über Ressourcen in die Ontologie zu schreiben bzw. diese zu lesen. Zu diesem Zweck besitzt der *Solver* einen *Resource Manager Proxy*. Dieser wird zur Kommunikation via JNI (Java Native Interface) mit einer in Java implementierten Applikation (*Resource Manager*) verwendet. Diese ist in der Lage, Daten mit der Ontologiedatenbank auszutauschen.

Abbildung 84 zeigt anhand eines Ablaufdiagramms das Hinzufügen von Perzeptionsmodulen in die Ontologie. Hierzu ist die Kommunikation zwischen dem *Processing Coordinator*, dem *Perception Solver*, dem *Resource Manager* und der *Wissensdatenbank* visualisiert. In dem Beispiel wird der *Processing Coordinator* erst gestartet, nachdem der *Perception Solver* initialisiert ist. Nachdem der *Perception Solver* gestartet wurde, initialisiert er über den *Resource Manager*

Proxy die in Java implementierte Applikation (*Resource Manager*). Diese wiederum baut eine Verbindung zur Datenbank auf. Nach dem Aktualisieren der Datenbank wird der *Solver* über die erfolgreiche (im Falle eines Fehlers, über die nicht erfolgreiche) Initialisierung des *Resource Manager* informiert. Beim Start des *Processing Coordinator* überträgt dieser Informationen der vorhandenen Module an den *Solver*. Dieser leitet die Module als OWL Individuen an den *Resource Manager*. Anschließend wird die Inferenzmaschine ausgeführt. Der Zustand der Ontologie wird aktualisiert und an den *Resource Manager* übertragen. Dieser informiert den *Solver* anschließend über die nun zur Verfügung stehenden Fähigkeiten (z.B. „detektieren“, „Fahrzeug“). Der *Solver* wiederum bestätigt dem *Coordinator* das erfolgreiche Hinzufügen der Module.

Dasselbe Vorgehen wird für alle *Clients* durchgeführt, sodass nach der Initialisierung aller Clients die einzelnen Individuen sowohl in der Datenbank stehen als auch der Solver Informationen darüber besitzt, welche Ressourcen von Clients bereitgestellt wurden und welche perzeptiven Fähigkeiten damit vorhanden sind. Das Vorhandensein von perzeptiven Fähigkeiten wird anschließend über die APCOM kommuniziert. Erhält der *Solver* über die APCOM den Befehl, eine bestimmte Fähigkeit auszuführen, durchsucht er die in Frage kommenden Perzeptionsketten, bewertet diese (nicht Teil dieser Forschungsarbeit, siehe [61]) und wählt die geeignetste Kette aus.

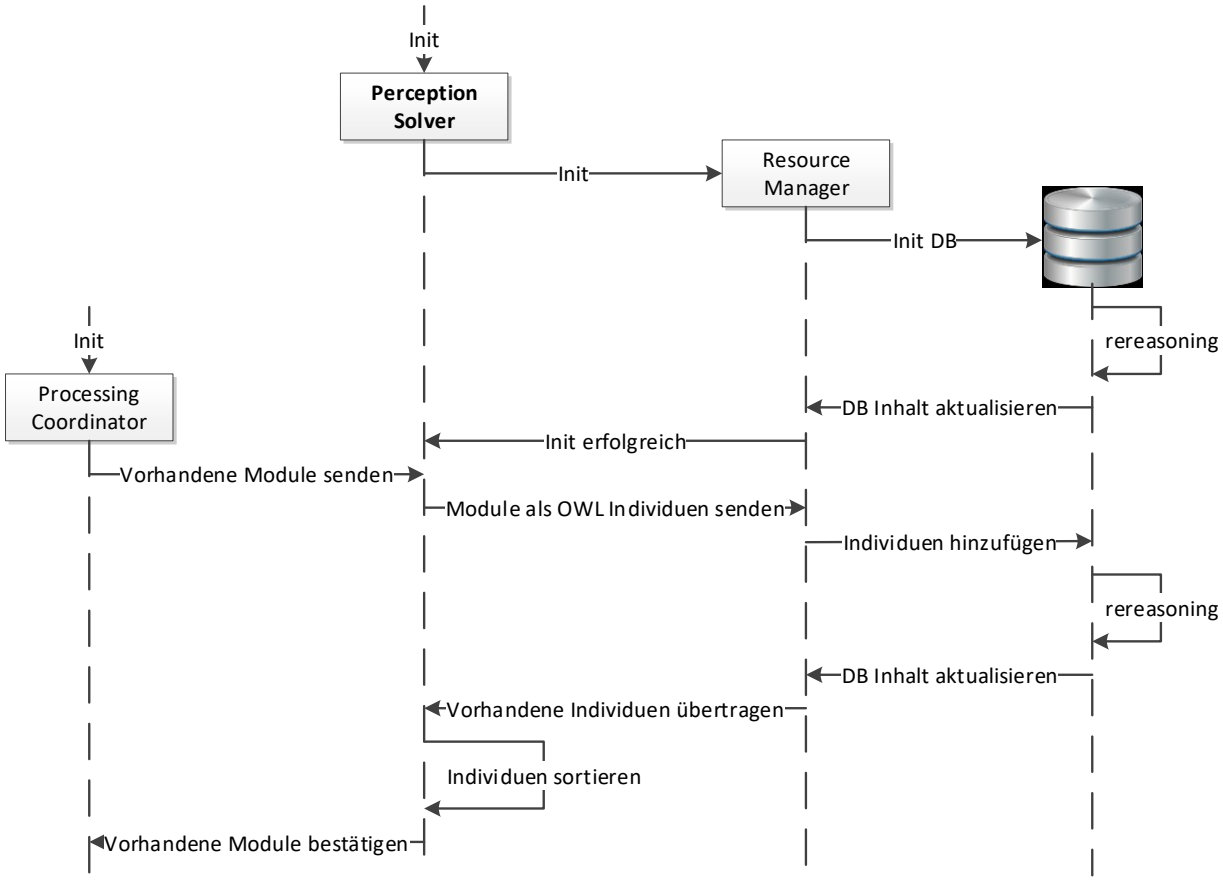


Abbildung 84: Ablaufdiagramm der Kommunikation zwischen Processing Coordinator, Perception Solver und dem Resource Manager zum Hinzufügen von Perzeptionsmodulen.

Die Module dieser Kette werden zum Ausführen an den *Coordinator* übertragen. Sobald der *Coordinator* dem *Solver* mitteilt, dass die Module erfolgreich gestartet wurden, informiert der *Solver* über die APCOM, dass die Aufgabe nun durchgeführt wird. Soll eine Aufgabe beendet werden, wird dies ebenfalls über die APCOM mitgeteilt. Der *Solver* leitet diese Information an den *Coordinator* weiter, und sobald der *Coordinator* das erfolgreiche Beenden der Module signalisiert hat, informiert der *Solver* über die APCOM, dass die Aufgabe erfolgreich beendet wurde und nicht weiter ausgeführt wird. Neben der Verwaltung der Ressourcen und dem Starten bzw. Stoppen von Perzeptionsketten ist der Solver auch für das Parametrisieren der Ressourcen zuständig. Beispielsweise kann dem Solver von außen mitgeteilt werden, mit welchen Parametern ein Sensor benutzt werden soll.

6.5 Umsetzung des Resource Managers

Im vorhergehenden Kapitel wurde bereits angesprochen, dass der *Solver* nicht in der Lage ist, direkt mit der Datenbank zu kommunizieren. Stattdessen wird der in Java implementierte *Resource Manager* zur Kommunikation mit der Datenbank genutzt. Der Austausch zwischen *Solver* und *Resource Manager* findet über den *Resource Manager Proxy* statt. Dieser implementiert zum Datenaustausch das JNI (Java Native Interface). Der Grund für diese Lösung liegt im Ursprung von OWL begründet. OWL wurde für das Semantic Web entwickelt, weshalb die „State oft the Art“ Bibliotheken in „Web-kompatiblen“ Sprachen, wie z.B. Java, geschrieben sind. Da es für OWL (und Ontologiedatenbanken bzw. Tripelstores) nur wenig[31] C++ Unterstützung in Form von Bibliotheken gibt und die Java-basierte OWL-API [131] als quasi Standard zur Entwicklung von OWL basierten Applikationen herangenommen wird, wurde die Applikation zur Kommunikation mit der Datenbank in Java realisiert. Es wurde darauf geachtet, möglichst wenig Funktionalität in Java zu implementieren, da das Gesamtframework (S&PM) bereits vor der Konzepterweiterung in C++ realisiert wurde. Dies ist sinnvoll, da es sich bei den meisten Applikationen des S&PM um hardwarenahe Software handelt und eine bessere Unterstützung in Form von Bibliotheken seitens C++ besteht. Zusätzlich bieten hardwarenahe Programmiersprachen gerade im Bildverarbeitungsbereich einen Performancevorteil. Der Datenaustausch zwischen den zwei Welten (Java und C++) wurde auf Basis des JNI realisiert.

Im folgenden Kapitel soll die Implementierung sowohl der Java Applikation (*Resource Manager*) als auch der C++ Applikation (*Resource Manager Proxy*) näher betrachtet werden. Der *Resource Manager Proxy* ist als Teil des *Solvers* implementiert und startet sowohl die *Java Virtual Machine* (JVM) als auch die Applikation selbst, sobald der *Solver* initialisiert wird.

6.5.1 Resource Manager Proxy (C++)

Der Resource Manager Proxy ist verantwortlich für zwei Aufgaben: (1) Das Initialisieren der *Java Virtual Machine* sowie das Starten des *Resource Managers* und (2) dient er als Datencon-

[31] Gut dokumentierte, getestete, zuverlässige und fehlerfreie C++ Bibliotheken

tainer für Informationen, die aus der Ontologiedatenbank extrahiert werden. Neben dem Initialisieren der JVM muss sichergestellt werden, dass der *Resource Manager* gestartet werden kann und alle Signaturen der Klassen und Methoden korrekt sind. Der *Resource Manager Proxy* hat dieselben Klassenstrukturen und Attribute wie der *Resource Manager*. Dies ist notwendig, um mittels JNI Daten aus Java nach C++ zu konvertieren. Jedoch sind die Logiken und Algorithmen (z.B. zur Interpretation und Konvertierung der Daten aus der Datenbank) komplett Javaseitig implementiert. D.h., die Klassen im *Resource Manager Proxy* sind reine Datencontainer.

In Abbildung 84 wurde bereits gezeigt, wie ein Perzeptionsmodul der OWL Ontologie hinzugefügt wird. Abbildung 85 veranschaulicht diesen Ablauf im Detail und zeigt auf, welche Schritte im *Resource Manager Proxy* durchgeführt werden müssen, um ein Individual der Datenbank hinzuzufügen. Die *Resource Manager Proxy* Klasse besitzt einen eigenen JNI Thread, der auf Anfragen wartet. Möchte der *Solver* ein Individuum hinzufügen, so wird das Individuum an den *Resource Manager Proxy* gesendet. Dieser informiert den JNI Thread, welche Operation mit welchen Daten als nächstes durchgeführt werden soll. Anschließend wartet der *Resource Manager Proxy* auf das Feedback des Threads. Sobald dem Thread mitgeteilt wird, welche Operation durchgeführt werden muss, schickt dieser ein JNI Querry an den *Resource Manager*. Der *Resource Manager* fügt die Daten der Ontologie hinzu bzw. extrahiert diese aus der Ontologie und informiert den Thread, dass die Operation erfolgreich durchgeführt wurde. Daraufhin informiert der JNI Thread den *Resource Manager Proxy*, dass die Operation abgearbeitet wurde, woraufhin auch dem *Solver* mitgeteilt wird, dass der Vorgang abgeschlossen ist.

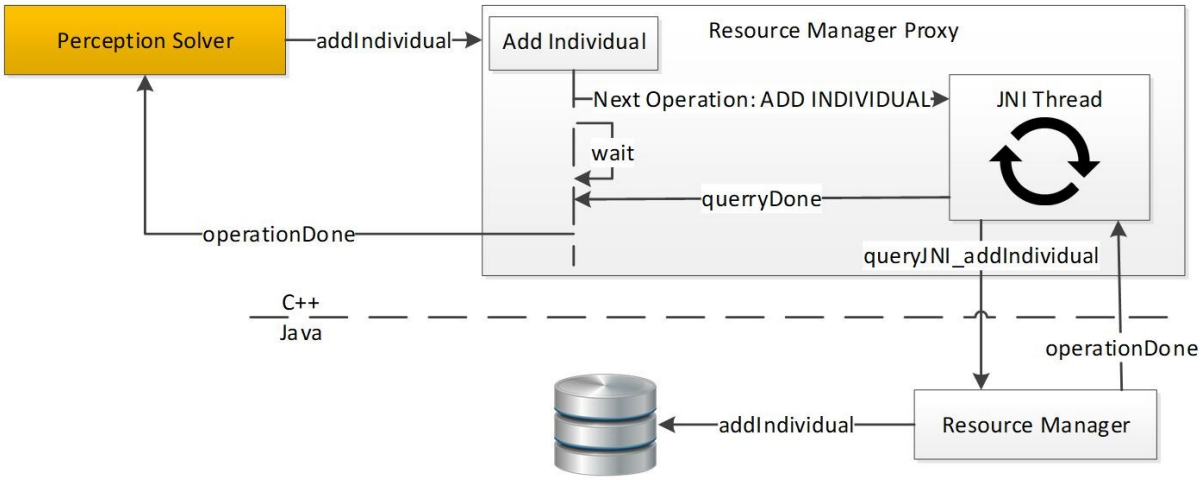


Abbildung 85: Funktionsablauf für das Hinzufügen eines Individuums vom *Perception Solver* über den *Resource Manager Proxy* bis zum *Resource Manager*.

Der *Resource Manager Proxy* verfügt über die folgenden Operationen:

- Hinzufügen bzw. Löschen von Individuen,
- Hinzufügen bzw. Löschen von abstrakten Rollen,
- Hinzufügen bzw. Löschen von konkreten Rollen,
- Speichern der in der Datenbank enthaltenen Ontologie in eine Datei,

- Bedienen der Inferenzmaschine,
- Auslesen aller Ressourcen,
- Auslesen aller Fähigkeiten,
- Auslesen von Konfigurationen und Aufgaben.

6.5.2 Resource Manager (Java)

Der *Resource Manager* wurde auf zwei verschiedene Arten implementiert, wobei die Schnittstellen identisch sind, damit beide Implementierungen kompatibel zum *Resource Manager Proxy* sind. Die erste Implementierung nutzt die Referenzimplementierung der OWL API[32], um Daten aus der Ontologie zu lesen bzw. in die Ontologie zu schreiben. Die OWL API ist eine Open Source Java Bibliothek, um OWL Ontologien zu erstellen und zu manipulieren. Die API inkludiert verschiedene Parser (RDF/XML, OWL/XML, Turtle, etc.) und unterstützt die folgenden Inferenzmaschinen: FaCT++ [132], Hermit [133], Pellet [134] und Racer [135]. Die zweite Implementierung nutzt die Stardog[33] Bibliotheken, um mit der gleichnamigen Tripelstore Datenbank zu kommunizieren. Während bei der Referenzimplementierung direkt auf einer „*.owl*“ Datei gearbeitet wird, d.h. die Tripel aus der Ontologiedatei in den Speicher geladen werden, wird bei Stardog die Ontologie in einer Datenbank gespeichert. Der Zugriff auf die Axiome erfolgt über die Abfragesprache SPARQL (vgl. Kapitel 5.3.2.4).

Im Folgenden werden die wichtigsten Querry erläutert. Alle Dataproperties samt ihrer Range Restriktionen können mittels

```
"SELECT ?p ?range WHERE {?p a owl:DatatypeProperty . ?p rdfs:range ?range  }"
```

ausgelesen werden. Um ein bestimmtes Objectproerty eines Individuals zu erhalten, verwendet man das folgende Querry:

```
"SELECT ?v WHERE { <individualName> ap:<propertyName> ?v .}"
```

`IndividualName` und `propertyName` sind Platzhalter für die entsprechenden Bezeichner. Mit den Querries

```
"SELECT ?p ?v  WHERE {  <individualName> ?p ?v . ?p a owl:DatatypeProperty }"
```

```
"SELECT ?p ?v  WHERE {  <individualName> ?p ?v . ?p a owl:ObjectProperty }"
```

erhält man alle Dataproperties bzw. Objectproperties eines Individuums. Möchte man sich die Instanzen einer bestimmten Klasse ausgeben lassen, wird das folgende Querry benutzt:

```
"SELECT ?x WHERE { ?x rdf:type ap:<klassenNamen>}"
```

Sobald alle notwendigen Informationen mittels der Querrys extrahiert wurden, werden die Perzeptionsketten mit dem in Abbildung 86 dargestellten Algorithmus generiert. Im oberen Teil

[32] http://owlapi.sourceforge.net/
[33] http://www.stardog.com/

ist die Reihenfolge der rekursiven Aufrufe zu sehen, darunter die Kettenzusammenstellungen zu den jeweiligen Zeitpunkten. Ganz unten wird die Ablaufreihenfolge zur Verdeutlichung noch einmal mittels annotierter Pfeile dargestellt.

Zunächst beginnt der Algorithmus bei R0. Von hier aus finden rekursive Aufrufe statt, bis der Algorithmus bei R4 stehen bleibt. Modul A wird als Teil einer Kette gespeichert und, nachdem der rekursive Aufruf R4 terminiert, befindet sich der Algorithmus in R3. In R3 wird der Kette aus R4 das Modul B hinzugefügt, sodass die Kette nun aus den Modulen A und B besteht. Nach dem Terminieren befindet sich der Algorithmus in R2. An dieser Stelle wird der bestehenden Kette das Modul E hinzugefügt, sodass sich die Kette nun aus den Modulen A, B und E zusammensetzt. Da das Modul sowohl von B als auch von D abhängig ist, folgen wieder rekursive Methodenaufrufe, bis der Algorithmus in R6 terminiert. Modul C markiert wieder einen Kettenanfang, woraufhin die eben beschriebenen Schritte erneut durchlaufen werden. Zum Zeitpunkt von R7 existieren zwei Ketten (A, B, E) und (C, D, E). Nachdem R7 terminiert ist, werden beide Ketten nach R1 übergeben und das Modul F angehängt.

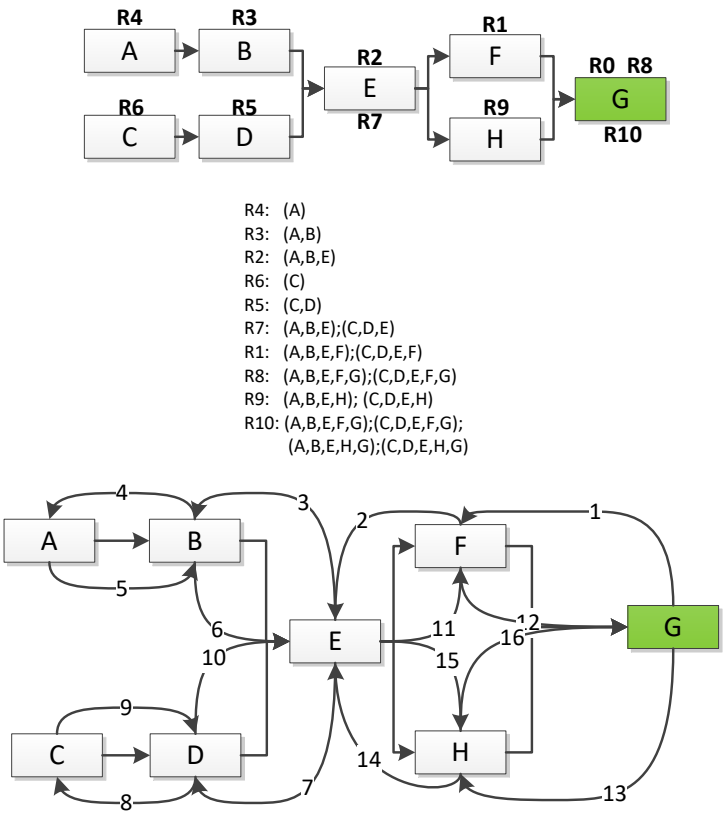


Abbildung 86: Rekursive Aufrufe für eine Perzeptionskette

Somit bestehen folgende zwei Ketten zum Zeitpunkt R1: (A, B, E, F) und (C, D, E, F). Der rekursive Aufruf R1 terminiert und übergibt beide Ketten an den Aufruf R8. Da R8 nun wiederum zwei Modulabhängigkeiten besitzt, findet wieder ein rekursiver Aufruf (R9) statt. Da die Ketten in Modul E bereits bekannt sind, müssen die rekursiven Aufrufe nicht erneut bis zum Anfang einer Kette durchlaufen werden, sondern können die vorhandenen Informationen in E benutzen. In R10 existieren somit vier Ketten, wie in der Abbildung dargestellt.

6.6 Umsetzung des Processing Coordinator

Der *Processing Coordinator* ist für das Verwalten, Ausführen und Konfigurieren der Perzeptionsketten zuständig. Neben dem Starten und Stoppen der einzelnen Module überwacht er auch deren Funktion und gibt eine Rückmeldung über Fehlfunktionen. Die Information über die zu startenden Module, samt ihrer Parameter, erhält der *Coordinator* in Form einer Perzeptionskette vom *Perception Solver*. Die gewählte Perzeptionskette gibt an, welche Module in welcher Reihenfolge ausgeführt werden müssen. Die Ergebnisse der Perzeptionsketten werden an den *Result Validator* übermittelt. Die Informationen zu den verschiedenen Modulen bezieht der *Coordinator* aus der *Module Database* (siehe Abbildung 21). Der *Coordinator* besteht aus zwei Teilen, deren Implementierung im Folgenden erläutert wird:

- die **Modulverwaltung** und Kommunikation: Zuständig für die Kommunikation mit z.B. dem *Perception Solver* und die Verwaltung der Module respektive Ketten.
- die **Module und ihre Ausführung**: Die Implementierung der einzelnen Module und deren Datenaustausch untereinander.

6.6.1 Die Modulverwaltung

Zum Systemstart interpretiert der *Coordinator* eine XML Datei mit Initialisierungsparametern, welche z.B. die Kommunikationsdaten zum *Perception Solver* und einen Systempfad zu spezifischen Parametern für unterschiedliche Module enthalten. Aus der Implementierung der Module ist der *Coordinator* in der Lage, Informationen über alle vorhandenen Module abzuleiten und diese an den *Solver* zu propagieren, wie bereits in den vorhergehenden Kapiteln erläutert wurde. Zusätzlich kann der *Solver* beim *Coordinator* die aktuell vorhandenen Module samt ihrer Parameter anfragen. Für jedes Modul existiert eine standardisierte Basiskonfiguration. Ist diese Basiskonfiguration für ein bestimmtes Modul nicht erwünscht, können spezifische Konfigurationsparameter und Eigenschaften über eine speziell für dieses Modul angelegte XML Datei herangezogen werden. Dies soll die Flexibilität beim Konfigurieren der Module erhöhen und es ermöglichen, die Konfiguration der Module außerhalb des Quellcodes anzupassen. Die Kommunikation mit dem *Coordinator* findet über ein Interface, dem *Coordinator Controller* statt. Verschiedene Clientimplementierungen können von diesem erben, um die Kommunikation mit dem *Coordinator* auf verschiedene Arten zu realisieren (vgl. Abbildung 87). Die aktuelle Implementierung sieht drei Klienten vor: Den *Perception Solver*, die Kommunikation mit dem *Coordinator* über ein Terminal und eine GUI, über die der *Coordinator* überwacht und manipuliert werden kann. Die interne Kommunikation zwischen *Coordinator Controller* und dem *Processing Coordinator* basiert auf einem Signal-Slot Prinzip. Empfängt der Controller ein Signal von außen, wird der *Coordinator* darüber informiert. Möchte der *Coordinator* umgekehrt einen Status an den *Controller* zurückgeben, wird im *Controller* ein Signal ausgelöst, das gleichzeitig alle Klienten über die Veränderung informiert. So wird gewährleistet, dass alle Klienten eine synchrone Datenbasis besitzen.

Erhält der *Controller* vom *Perception Solver* z.B. das Signal, eine Kette zu starten, werden die Informationen der Kette (z.B. ID) zusammen mit den zu startenden Modulen an den *Coordinator* weitergeleitet. Dieser prüft, ob die Kette gültig ist bzw. ob die Perzeptionsaufgabe mit derselben ID bereits existiert. Treten keine Fehler auf, wird eine neue Perzeptionsaufgabe mit der zu startenden Kette angelegt, und die notwendigen Module werden gestartet. Abbildung 88 veranschaulicht diesen Vorgang. Zunächst muss überprüft werden, ob der *Solver* eine tatsächlich existierende Ketten-ID gewählt hat und in einem zweiten Schritt, ob der *Coordinator* eine gültige Implementierung für das zu startenden Modul besitzt.

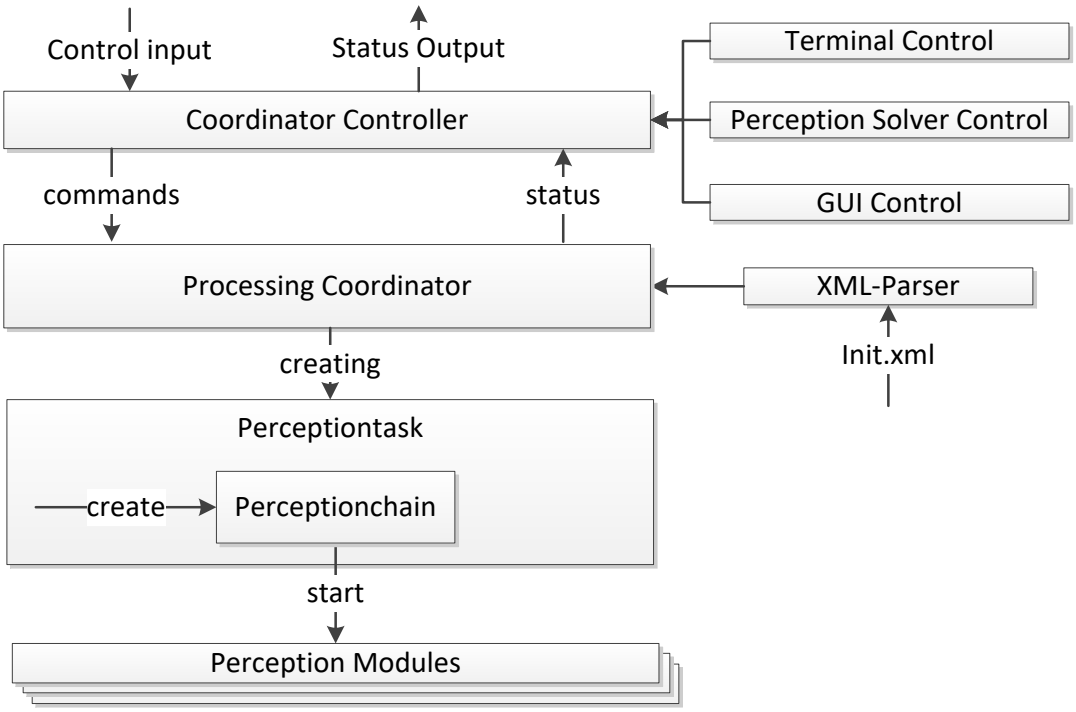


Abbildung 87: Processing Coordinator: Die Applikation

Wurden alle Konsistenz- und Integritätsüberprüfungen erfolgreich durchlaufen, wird intern eine neue Perzeptionsaufgabe (*Perceptiontask*) angelegt und anschließend für jedes Modul überprüft, ob dieses in der vorgegebenen Konfiguration bereits läuft. Ist dies der Fall, muss das Modul nicht noch einmal neu gestartet werden. Lediglich die Zuordnung zu mehreren *Perceptiontasks* (das Modul wird nun von mehr als einem Perzeptionstask verwendet) muss angepasst werden. Wird dieses Modul noch nicht ausgeführt, wird es der Perzeptionsaufgabe hinzugefügt. Im letzten Schritt wird der *Controller* darüber informiert, dass die Kette erfolgreich gestartet wurde.

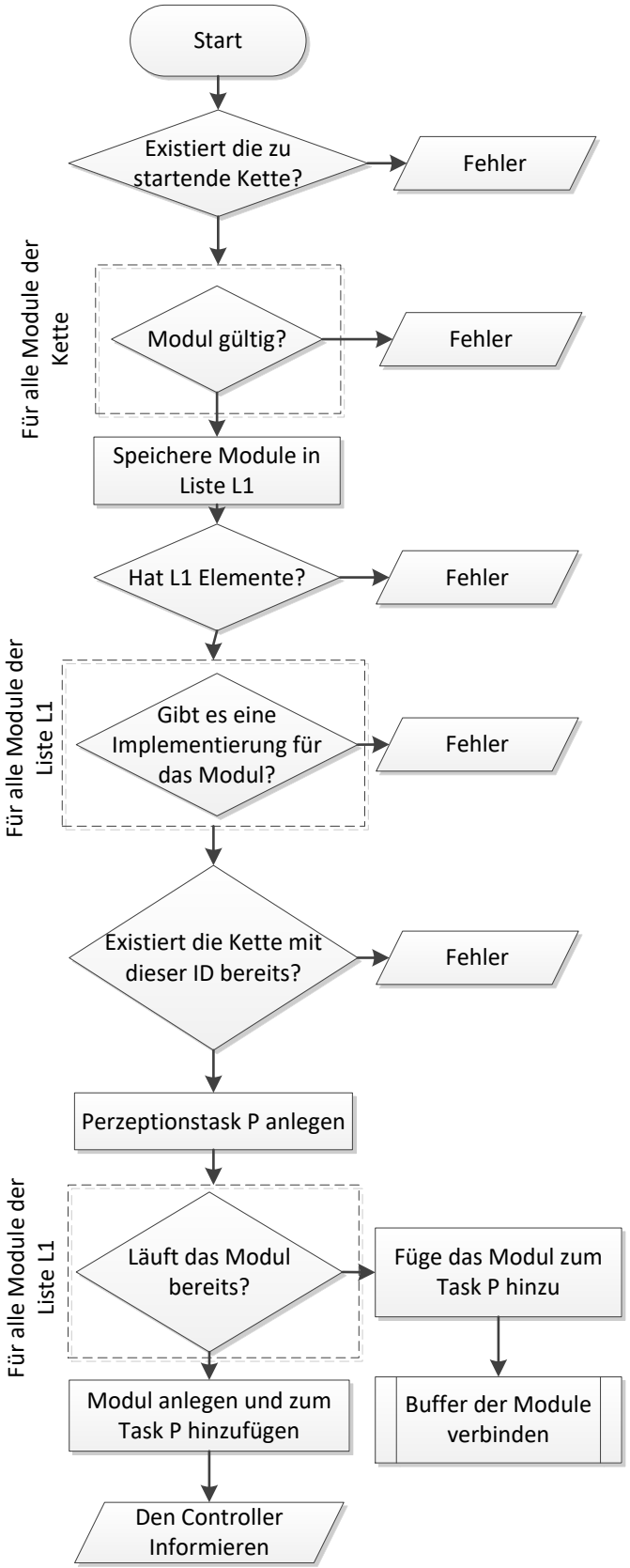


Abbildung 88: Anlegen eines Perzeptionstasks und Starten der notwendigen Perzeptionsmodule

Als nächstes soll nun der Algorithmus zum Anlegen eines Moduls näher betrachtet und dessen Buffer Zuordnung erläutert werden. Abbildung 89 veranschaulicht diesen Vorgang näher.

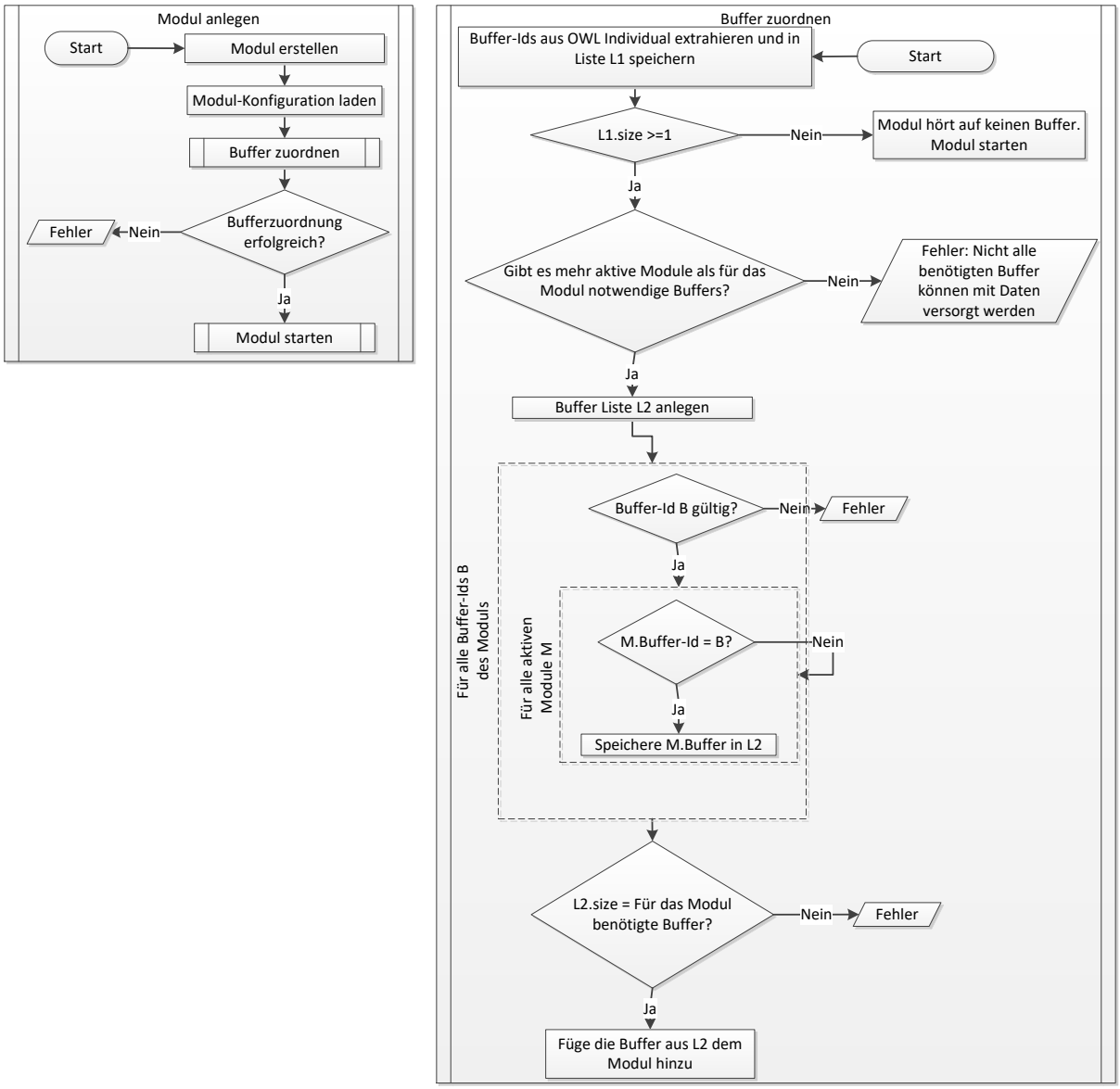


Abbildung 89: Algorithmus zum Anlegen eines Moduls und Zuweisung der notwendigen eingehenden Buffer

Als erstes wird anhand der Modul-ID (vgl. Abbildung 91) ermittelt, welches Modul erstellt werden soll. Anschließend werden anhand des OWL Individuums die Konfigurationsparameter für das Modul ermittelt und dem Modul zugewiesen. Im nächsten Schritt werden die notwendigen Buffer, über welche die Daten bezogen werden, dem Modul zugeordnet. Ist die Buffer Zuordnung erfolgreich, wird das Modul gestartet. Im anderen Fall wird ein Fehler ausgegeben und sowohl der *Coordinator* als auch der *Solver* und andere Klienten werden darüber informiert, dass ein Modul nicht erfolgreich gestartet werden konnte. Dies führt dazu, dass die Perzeptionsaufgabe nicht ausgeführt werden kann.

Die Buffer Zuordnung ist notwendig, damit ein Modul weiß, welche Daten von welchem Vorgängermodul bezogen werden sollen (vgl. Abbildung 90). Dies setzt voraus, dass bereits Module aktiv sind, die einen Buffer, von dem Daten bezogen werden können, zur Verfügung stellen. Zunächst werden die Buffer-Ids aus dem zugehörigen OWL Individuum extrahiert und in

einer Liste L1 gespeichert. Ist diese Liste leer, bedeutet es, dass das Modul keine Daten von einem Vorgängermodul bezieht. Dies trifft für die Module zu, die ihre Daten von einer anderen externen Quelle, z.B. Sensoren oder über das Netzwerk, beziehen. Das Modul kann somit ohne eine Buffer Zuordnung gestartet werden. Enthält die Liste L1 mindestens ein Element, wird als nächstes überprüft, wie viele Buffer ein Modul benötigt (ein Modul kann die notwendigen Daten von mehreren Buffern, stellvertretend Modulen beziehen) und ob genug aktive, d.h. laufende Module vorhanden sind. Ist dies nicht der Fall, können dem Modul nicht alle Daten zur Verfügung gestellt werden, die es benötigt, und das Modul kann nicht gestartet werden. Gibt es genug aktive Module, wird eine Liste L2 für die Buffer angelegt und für alle Buffer-Ids B aus der Liste L1 überprüft, ob die Buffer-Ids a) gültig sind und b) die Buffer-Id B identisch ist mit einer Buffer-Id der bereits aktiven Module. Ist dies der Fall, bedeutet es, dass ein aktives Modul gefunden wurde, dessen Daten für die Verarbeitung herangezogen werden können. Dieser Buffer wird in der Liste L2 gespeichert. Nachdem alle Buffer-Ids durchsucht und die gültigen Buffer in der Liste L2 gespeichert wurden, wird überprüft, ob die Liste L2 genug Elemente enthält. Für jedes Modul ist festgelegt, wie viele Buffer es benötigt. Stimmt diese Zahl mit der Zahl der Elemente in L2 überein, können die Buffer dem Modul zugewiesen und, sofern beim Zuweisen der Buffer kein Fehler aufgetreten ist, das Modul gestartet werden.

6.6.2 Module und ihre Ausführung

Jedes Modul ist Teil eines Bildverarbeitungsalgorithmus. Dieser reicht von einfachen Aufgaben wie beispielsweise einer Farbkonvertierung oder einer komplexen Abfolge von Verarbeitungsschritten bis zum Detektieren von Objekten in einem Bild. Jedes Modul wird dabei in einem eigenen Thread ausgeführt. Dies gewährleistet, dass jedes Modul unabhängig von anderen Modulen arbeiten kann und nicht warten muss, bis das Vorgängermodul die Daten prozessiert hat. Um Rechenleistung zu sparen, befindet sich ein Modul solange im Schlafzustand (*idle*), bis Daten vorhanden sind. Die Module sind untereinander über ein Buffersystem verbunden. Jedes Modul bezieht Daten von verschiedenen Buffer. Gleichzeitig speichert jedes Modul die eigenen Ergebnisse in einem Buffer (vgl. Abbildung 90). Sobald Daten in einen Buffer abgelegt wurden, wird eine Benachrichtigung an alle, die mit dem Buffer verknüpft sind, verschickt, um diese zu aktivieren.

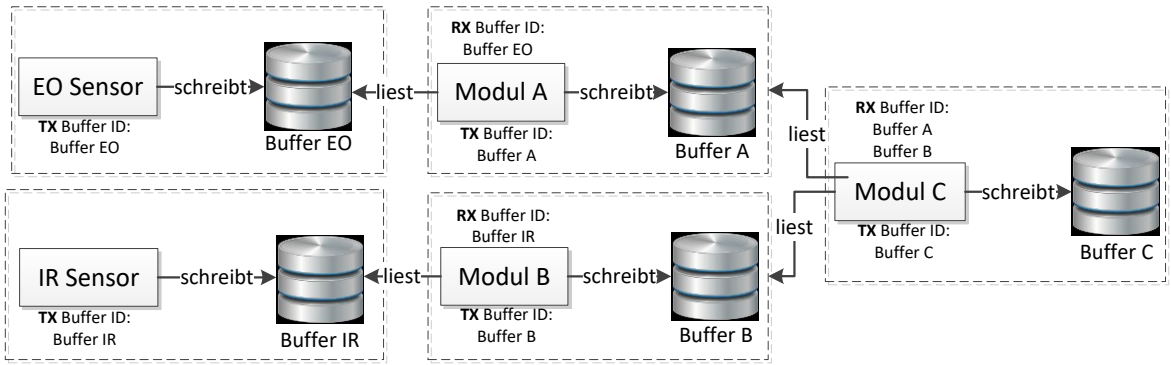


Abbildung 90: Verknüpfung von Modulen über einen Buffer

Für jedes Modul gibt es ein Basisinterface (*IModule*), von dem geerbt werden muss. Das Basisinterface enthält zwingend notwendige Attribute und Methoden, die für jedes Modul implementiert bzw. befüllt werden müssen. Die Klasse *IModule* ist in Abbildung 91 in Form eines Klassendiagramms zusehen.

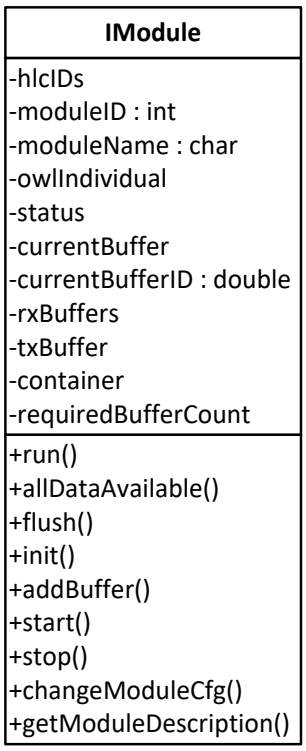


Abbildung 91: UML Klassendiagramm der Klasse IModule

Die Klasse besitzt folgende Attribute:

- *hlcIDs*: Ein Vektor mit Perceptiontask-IDs: Mit diesen IDs wird vermerkt, zu welchen Ketten respektive Perzeptionsaufgaben ein Modul gehört.
- *moduleID*: Die ID des Moduls.
- *moduleName*: Der Name des Moduls.
- *owlIndividual*: Jedem Modul ist ein stellvertretendes OWL Individuum zugeordnet, welches die Ontologiedaten enthält.
- *status*: Im status-Attribut wird der Fehlerstatus des Moduls vermerkt.
- *currentBuffer*: Ein Zeiger auf den Buffer, welcher vom Modul benutzt wird.
- *currentBufferID*: Die ID des Buffers.
- *rxBuffers*: Vektor von mehreren Inputbuffern (Ein Modul kann Daten aus mehreren Buffern beziehen).
- *txBuffer*: Der Buffer, in welchem das Modul seine Ergebnisse ausgibt.
- *container*: Die Containerklasse ist eine eigene Klasse, die zum Datenaustausch verwendet wird. Der Container enthält die zu verarbeitenden Daten (z.B.: ROIs oder Bildinformationen).

- *requiredBufferCount*: Um zu gewährleisten, dass ein Modul richtig funktioniert und mit allen notwendigen Buffer verbunden wird, existiert ein Attribut, welches festlegt, wie viele Buffer das jeweilige Modul benötigt.

In der *run*-Methode ist die Ablauflogik eines jeden Moduls implementiert. Dabei muss sichergestellt werden, dass alle notwendigen Daten vorhanden sind (z.B. wenn Daten aus mehreren Buffern benötigt werden). Hierfür wird die *allDataAvailable*-Methode aufgerufen, um zu überprüfen, ob bereits alle notwendigen Daten vorhanden sind. Mit der *addBuffer*-Methode wird sichergestellt, dass der richtige Buffer dem richtigen Slot zugeordnet wird und die Daten nicht vertauscht werden. *Init*, *flush*, *start* und *stop* werden aufgerufen, um den Thread zu kontrollieren. Aus der *getModuleDescription*-Methode wird die Basisbeschreibung eines Moduls ausgelesen. Diese Beschreibung wird genutzt, um daraus ein OWL-Individual anzulegen und es in der Ontologie zu speichern. Die *getModuleDescription*-Methode wird aufgerufen, sofern es keine spezifische XML-Datei mit einer alternativen Beschreibung für dieses Modul gibt. *ChangeModuleCfg* wird aufgerufen, wenn z.B. der *Solver* dem *Coordinator* mitteilt, dass sich während der Laufzeit des Moduls die Modulparameter ändern. Dies können Algorithmus-spezifische Parameter sein oder Parameter, die das Modul zur Kommunikation mit anderen Modulen bzw. Services, wie z.B. einer GIS-Datenbank, benötigt.

7

Evaluierung

Im Folgenden wurden das entwickelte Softwaresystem und die Ontologie hinsichtlich ihrer Struktur sowie ihrer operationellen Einsetzbarkeit und Leistung evaluiert und bewertet. Die Untersuchungen fanden in drei Bereichen statt:

1. Strukturelle Bewertung: Bewertung der Ontologie hinsichtlich ihrer internen Struktur (Klassenhierarchie, Data- und Objectproperties, Instanzen) und des Designs. Zur Bewertung wurden Vorgehensweisen aus dem *OntologySummit* 2013[34] verwendet.
2. Operationelle Bewertung: In verschiedenen Anwendungsfällen wurde evaluiert, ob und wie das Softwaresystem und die Ontologie die in Kapitel 3 genannten Anforderungen erfüllen.
3. Leistungsbewertung: Untersuchung der Zugriffs- bzw. Abfragezeiten innerhalb der Ontologie auf unterschiedliche Informationen im Rahmen der Aufgabenstellung. Hierbei wurden zwei verschiedene Ansätze verglichen.

Für die Untersuchungen wurde die in Kapitel 6.2 beschriebene Ontologie verwendet. Die Klassenhierarchie kann der Abbildung 48 entnommen werden. Die Beziehungen sind in Abbildung 65 dargestellt. Die Ontologie enthielt Instanzen (Details sind den Tabellen aus Kapitel 6.2.5 zu entnehmen) aus folgenden Klassen:

- 44 Instanzen der Klasse Capability,
- 5 Instanzen der Klasse Environment,
- 52 Instanzen der Klasse GenericDescription,
- 5 Instanzen der Klasse OperationMethod,
- 15 Instanzen der Klasse Entity,
- 15 Instanzen der Klasse Avionic,
- 11 Instanzen der Klasse Platform,
- 20 Instanzen der Klasse Software.

[34] Vgl. OntologySummit 2013, http://ontologforum.org/index.php/OntologySummit2013_Intrinsic_Aspects_Of_Ontology_Evaluation_Synthesis

7.1 Strukturelle Bewertung

Aufgrund fehlender Methoden für die Entwicklung bzw. eine konsequente Evaluierung einer Ontologie fand im Jahre 2013 das *Ontology Summit* zum Thema „Ontologiebewertung und Evaluierung“ statt (*OntologySummit2013 – „Ontology Evaluation Across the Ontology Lifecycle“*)[136],[137]. Ziel war es, geeignete Richtlinien festzulegen. Entstanden sind *best practices*, die bei der Entwicklung der vorliegenden Ontologie angewandt wurden. Ziel der strukturellen Validierung war es, eine Aussage über die Qualität der modellierten Ontologie zu erhalten. Es wurde untersucht, inwiefern das Design der Ontologie den gängigen Entwurfsmustern und *best practices* entspricht.

7.1.1 Untersuchungsgegenstand

Untersuchungsgegenstand waren Eigenschaften der Ontologie, die die domänenunabhängige Modellierung betreffen. Das heißt, es wurde untersucht, inwieweit Regeln eingehalten wurden, die beim Design einer beliebigen Ontologie berücksichtigt werden sollten. Zu diesen zählten unter anderem Abhängigkeiten und Verbindungen zwischen Konzepten, sowie Metaeigenschaften wie *Symmetrie*, *Reflexivität* und *Transitivität*. Damit sollte sichergestellt werden, dass beim Erstellen der Ontologie Schritte beachtet werden, die z.B. die Konsistenz der Ontologie wahren oder dass Konzepte sinnvoll (z.B. disjunkte Klassen) umgesetzt wurden. Die Untersuchung sollte helfen, eine Aussage darüber treffen zu können, ob und inwieweit Designstandards bzw. *best practices* genutzt und umgesetzt wurden.

Da es sich hier um allgemeine Regeln handelt, die auf jede Ontologie angewendet und nach einem festen Schema überprüft werden können, sind Werkzeuge entstanden, die diese Art von Tests automatisiert durchführen. In dem erwähnten *OntologySummit* wurden hierfür verschiedene Werkzeuge (z.B. *OntoClean* [138], [63], *OntoCheck* [109], *ODEval* [126] und viele weitere[96]) untersucht. Von diesen bekam der *OntOlogy Pitfall Scanner (OOPS!)* von María Poveda [81] den meisten Zuspruch. Dies liegt unter anderem daran, dass Poveda in ihrer wissenschaftlichen Ausarbeitung bereits viele vorhandene und bekannte Werkzeuge (u.a. die oben erwähnten) in ihre Arbeit einbezogen und untersucht hat. Entstanden ist ein Katalog von „häufig gemachten Fehlern“ (sogenannten *Pitfalls*), der genutzt werden kann, um zu zeigen, dass bei der Entwicklung einer Ontologie die *best practices* eingehalten wurden. Das Ziel ihrer Arbeit formulierte Poveda wie folgt:

> *„The main goals of this thesis are (a) to help ontology engineers to diagnose their ontologies in order to find common pitfalls and (b) to lessen the effort required from them by providing the suitable technological support”* [81]

Das Bewertungstool wurde zum Zeitpunkt der Publikation ihrer Arbeit bereits über 3000 Mal zum Evaluieren von Ontologien eingesetzt.

7.1.2 Methode und Experimente

Um die im vorhergehenden Abschnitt beschriebenen Eigenschaften zu evaluieren, wurden zwei Werkzeuge eingesetzt: Die Inferenzmaschine *Hermit* [133] und der in Kapitel 7.1.1 erwähnte *Pitfall Scanner*.

7.1.2.1 Kontinuierliche Verifikation mit Hermit

Hermit wurde von Beginn an kontinuierlich eingesetzt, um allgemeine Fehler, wie z.B. Inkonsistenzen in der Modellierung der Klassen, beim Design der Ontologie schon während der Entwicklung und beim Ausarbeiten der Konzepthierarchien zu verhindern.

Hermit unterstützt alle Eigenschaften von OWL2 einschließlich aller Datentypen. Durch Anwenden der mit diesem Werkzeug umgesetzten Inferenzregeln wurde aufgrund der Verknüpfung der verschiedenen Konzepte auf implizites Wissen geschlossen. Dadurch steigt die Anzahl der Axiome und die Komplexität der Ontologie. Durch den Einsatz der Inferenzmaschine wurde aber auch überprüft, ob *Data*- und *Objectproperties* eingehalten wurden (z.B. *Domain* und *Range* Beschränkungen). Außerdem ist *Hermit* in der Lage, strikt zwischen *Data*- und *Objectproperties* zu trennen bzw. diese zu klassifizieren. Ein Hauptpunkt ist die Nutzung des Hypertableau-Verfahrens [72], welches Axiome so übersetzt, dass Nichtdeterminismus möglichst vermieden wird (im Vergleich zum Tableau-Verfahren). Das Überprüfen von Konzepten innerhalb der Ontologie findet statt, indem Hermit Konsistenztests generiert. Es besitzt Inferenzregeln, die verwendet werden, um zu überprüfen, ob die in der Beschreibungslogik (vgl. Kapitel 5.1) festgelegten Regeln und Eigenschaften eingehalten werden. Findet ein Verstoß gegen diese Regeln statt (z.B., wenn eine *Range* Beschränkung verletzt wird), macht Hermit den Ontologiedesigner auf diesen Verstoß aufmerksam.

Für Instanzen einer Klasse wurde mittels der Disjunktheit (im Sinne von `owl:disjointWith`) überprüft und sichergestellt, dass Instanzen nicht zwei widersprüchlichen Klassen zugeordnet werden. Zum Beispiel wurde so gewährleistet, dass ein *Sensor* nicht gleichzeitig eine *Plattform* sein kann. Die Inferenzmaschine überprüft dafür folgende Regeln, die beispielhaft am Konzept *AerialPlatform* und dessen Unterklassen (siehe Abbildung 92) erläutert werden:

$$\text{Platform} \sqsubseteq (\text{AerialPlatform} \sqcup \text{SeaPlatform} \sqcup \text{UnderwaterPlatform} \sqcup \text{Groundplatform})$$

$$\text{AerialPlatform} \sqsubseteq (\text{Manned} \sqcup \text{Unmanned})$$

$$\text{Unmanned} \sqsubseteq (\text{EnduranceUAV} \sqcup \text{SmallUAV} \sqcup \text{TacticalUAV})$$

$$\text{EnduranceUAV} \sqsubseteq (MALE \sqcup \text{HALE})$$

$$\text{SmallUAV} \sqsubseteq (MiniUAV \sqcup \text{MicroUAV})$$

$$\text{TacticalUAV} \sqsubseteq (Wilga \sqcup \text{Pioneer} \sqcup \text{Hunter})$$

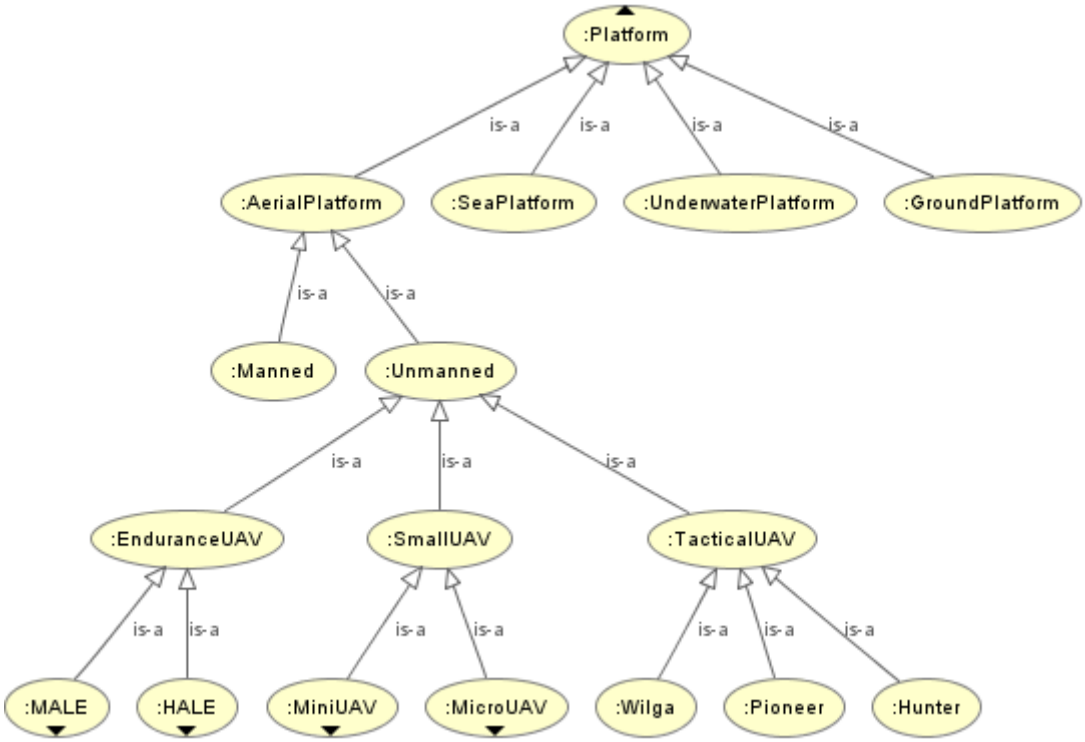


Abbildung 92: Taxonomie für das Konzept Platform

Die Überprüfung der Disjunktheit wurde nicht nur auf Klassen angewendet, sondern auch auf *Object*- und *Dataproperties*. Wird z.B. die UAV-Instanz *Predator-P1* sowohl der Klasse *AerialPlatform* als auch *UnderwaterPlatform* zugewiesen, erkennt Hermit das als Regelverstoß und liefert gleichzeitig eine Erklärung, warum diese Instanz nicht beiden Klassen gleichzeitig zugewiesen werden kann (vgl. Abbildung 93).

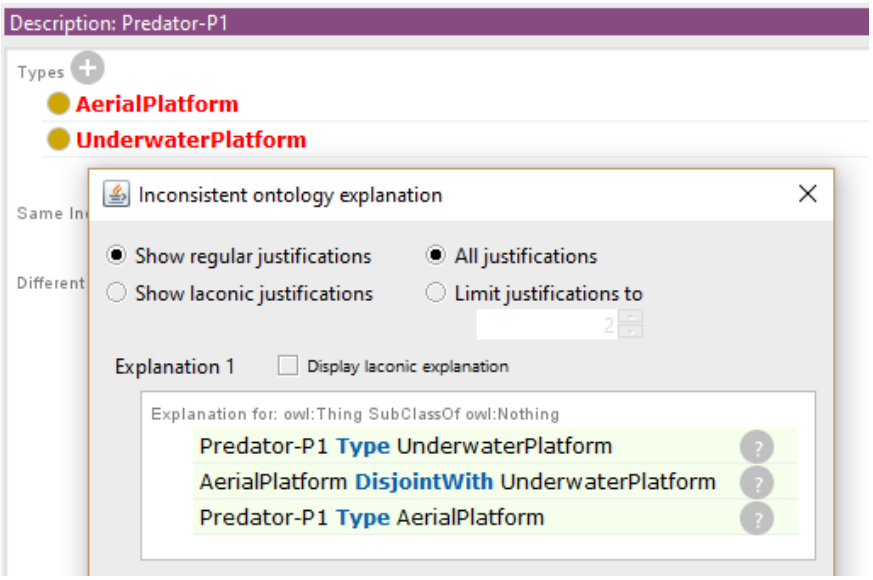


Abbildung 93: Regelverstoß bei der Zuweisung von zwei disjunkten Klassen

Mit den *Domain*- und *Range*-Restriktionen für *Objectproperties* wurde durch *Hermit* überprüft, ob eine Instanz aufgrund von falscher Zuordnung einer Beziehung oder über Vererbung einer ungültigen Klasse (vgl. Disjunktheit) zugewiesen wurde.

Dies wird mit der Beziehung „*providesCapability*“ verdeutlicht (siehe Abbildung 98).

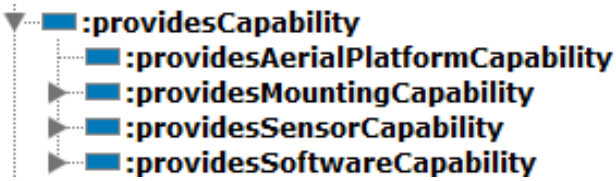


Abbildung 94: Die Beziehung „*providesCapability*“ und deren Unterbeziehungen

Die Beziehung „*providesCapability*“ soll zum Ausdruck bringen, dass eine Instanz eine Fähigkeit zur Verfügung stellt. Ohne eine Angabe von *Range*- und *Domain*-Restriktion gäbe es die Möglichkeit, jede Instanz mit jeder anderen Instanz zu verbinden. Damit wäre das Anlegen des folgenden unsinnigen Axioms möglich:

$$providesCapability(Sensor1, UAV1)$$

Durch Angabe einer *Range*-Restriktion für „*providesCapability*“

$$\exists providesCapability.Capability$$

wurde sichergestellt, dass das Objekt des Tripels der Klasse *Capability* zugeordnet wurde. Somit können aus dem vorhergehenden Beispiel zwei Fälle eintreffen:

1. Sind die Klassen *UAV* und *Capability* nicht disjunkt, wird die Instanz UAV1 aufgrund der Beziehung „*providesCapability*“ der Klasse *Capability* zugeordnet. Der Inferenzmaschine ist bekannt, dass das Objekt eines Tripels der Klasse *Capability* angehören muss, sie zieht daher den Schluss, dass UAV1 der Klasse *Capability* angehört.
2. Sind die Klassen *UAV* und *Capability* disjunkt, erkennt die Inferenz einen Fehler. UAV1 gehört bereits der Klasse *UAV* an, muss jedoch aufgrund der Definition von „*providesCapability*“ ebenfalls der Klasse *Capability* angehören. Da diese zwei Klassen jedoch disjunkt sind, ist die Ontologie nicht konsistent und es wird ein Fehler gemeldet.

Mit der Einführung einer *Range*-Restriktion wurde sichergestellt, dass das Objekt des Tripels korrekt zugeordnet werden konnte. Damit können folgende Axiome formuliert werden:

$$providesCapability(Sensor1, senseVisibleLight)$$

$$providesCapability(Sensor1, hover)$$

In dem oberen Beispiel sind beide Objekte des Tripels Instanzen der Klasse *Capability*. Während jedoch das erste Beispiel korrekt ist (ein Sensor kann sichtbares Licht wahrnehmen), macht das zweite Beispiel keinen Sinn (ein Sensor kann nicht schweben). Um solche Fehler zu vermeiden, ist eine *Domain*-Restriktion notwendig. Diese legt fest, welcher Klasse das Subjekt eines Tripels angehören darf. Um diese Aussagen trennen zu können, wurde in Abbildung 94 die Beziehung „*providesCapability*“ in Sub-Beziehungen untergliedert. Die Beziehung „*providesSensorCapability*“ erbt von „*providesCapability*“, womit festgelegt wird, dass das Objekt

des Tripels der Klasse *Capability* angehören muss. Zusätzlich wird die Domain dieser Beziehung der Klasse *Sensor* zugeordnet, sodass gilt:

$$\forall X\ \forall Y: Sensor(X,Y)\ \leftrightarrow (\text{providesSensorCapability}(X,Y) \wedge Capability(Y))$$

So wird sichergestellt, dass die Beziehung „*providesSensorCapability*“ nur in Verbindung mit Instanzen der Klasse *Sensor* und *Capability* benutzt werden kann.

Domain- und *Range*-Restriktionen wurden nicht nur für *Objectproperties*, sondern auch für *Dataproperties* festgelegt. Dadurch konnte überprüft werden, ob der Wert eines *Dataproperty* einem bestimmten Datentyp angehörte (vgl. Abbildung 95).

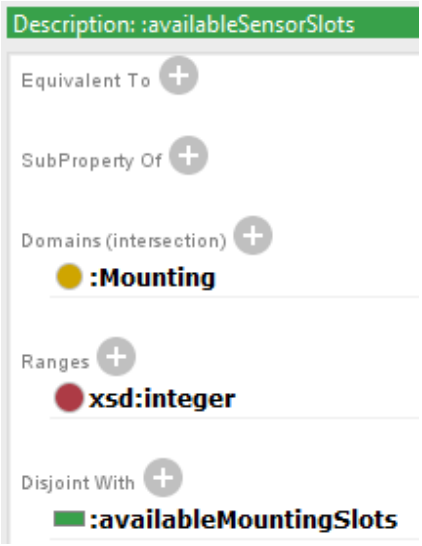


Abbildung 95: Beispiel für die Restriktionen eines *Dataproperty*

In Abbildung 95 wurde für das *Dataproperty* „*availableSensorSlots*“ festgelegt, dass das Subjekt eines Tripels der Klasse *Mounting* angehören und der Wert einer Integer-Zahl entsprechen muss. Zusätzlich wurde festgelegt, dass „availableSensorSlots“ und „availableMountingSlots“ disjunkt sind. Hierdurch war die Inferenzmaschine in der Lage zu prüfen, ob Datenwertzuweisungen von Instanzen korrekt waren.

7.1.2.2 Abschlussverifikation mit dem Pitfall Scanner

Während der kontinuierliche Einsatz der Inferenzmaschine sicherstellte, dass die Ontologie konsistent ist, bietet sie jedoch keine Möglichkeit, Designregeln und *best practices* zu prüfen. Werden z.B. für eine Beziehung Restriktionen angegeben, kann die Inferenzmaschine zwar überprüfen, ob diese Restriktionen eingehalten wurden, nicht jedoch kontrollieren, ob für alle Beziehungen Restriktionen festgelegt wurden. Für solche Fälle wird der *Pitfall Scanner* eingesetzt. Er weist den Designer auf mögliche Fehler hin. Die Entscheidung, diese Fehler zu beseitigen oder zu ignorieren, obliegt dem Designer. Ein Beispiel für die Auswertung des Scanners zeigt Abbildung 96.

Results for P21: Using a miscellaneous class.	**1 case | Minor**
Results for P22: Using different naming conventions in the ontology.	**ontology* | Minor**
Results for P27: Defining wrong equivalent properties.	**1 case | Critical**

Two object properties or two datatype properties are defined as equivalent, using owl:equivalentProperty, even though they do not have the same semantics.

• This pitfall appears in the following elements:
› http://code.ifs.lrt.unibw-muenchen.de/files/public/Fuzzy.owl#precision may not be equivalent to http://unibw.de/owl#precision

Results for P30: Equivalent classes not explicitly declared.	**1 case | Important**

This pitfall consists in missing the definition of equivalent classes (owl:equivalentClass) in case of duplicated concepts. When an ontology reuses terms from other ontologies, classes that have the same meaning should be defined as equivalent in order to benefit the interoperability between both ontologies.

• The following classes might be equivalent:
› http://unibw.de/owl#Detection, http://unibw.de/owl#Sensing

Abbildung 96: Beispielergebnis des *Pitfall* Scanners

Zu sehen sind verschiedene *Pitfalls,* die in der Ontologie identifiziert wurden. Zusätzlich wird angegeben, wie häufig diese aufgetreten sind und zu welcher Kategorie sie gehören. Zu jedem *Pitfall* können detaillierte Informationen eingeholt werden, z.B. darüber, an welcher Stelle ein Fehler aufgetreten ist.

Die Ontologie wurde, nachdem alle Konzepte implementiert und beschrieben wurden, mit dem *Pitfall Scanner* überprüft. Das automatisierte Überprüfen der Ontologie wurde unter http://oops.linkeddata.es/ durchgeführt. Trotzdem ist das erfolgreiche Beseitigen aller *Pitfalls* kein Garant dafür, dass die Ontologie zu 100% korrekt ist. Poveda merkt an, dass nicht immer alle *Pitfalls* erkannt werden können:

> *„… will not cover the cases in which some background and common sense knowledge is needed. For example, the pattern will not flag a pitfall when in a math ontology, the relationship "lessThan" is defined as inverse of "greaterThanOrEqual"… There might be cases in which developers leave domains and ranges open in order to foster interoperability, leaving behind precision in definition."* [81]

Für den Test wurden alle Pitfalls des Scanners genutzt. Zum Zeitpunkt der Erstellung dieser Arbeit existierten 41[35] *Pitfalls,* die in fünf Kategorien zusammengefasst wurden:

1. *Human understanding,*
2. *Logical consistency,*
3. *Real world representation,*
4. *Modelling issues,*
5. *Ontology language specification* (siehe Abbildung 97).

[35] Der Katalog wird ständig erweitert. Es besteht die Möglichkeit, Vorschläge für weitere Pitfalls einzureichen.

Human understanding	Modelling issues
• P1. Creating polysemous elements • P2. Creating synonyms as classes • P7. Merging different concepts in the same class • P8. Missing annotations • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P19. Swapping intersection and union • P20. Misusing ontology annotations • P22. Using different naming criteria in the ontology • P30. Missing equivalent classes • P32. Several classes with the same label	• P2. Creating synonyms as classes • P3. Creating the relationship "is" instead of using "rdfs:subClassOf", "rdf:type" or "owl:sameAs" • P4. Creating unconnected ontology elements • P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P7. Merging different concepts in the same class • P10. Missing disjointness • P17. Specializing too much a hierarchy • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P14. Misusing "owl:allValuesFrom" • P15. Misusing "not some" and "some not" • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P21. Using a miscellaneous class • P23. Using incorrectly ontology elements • P24. Using recursive definition • P25. Defining a relationship inverse to itself • P26. Defining inverse relationships for a symmetric one • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships • P30. Missing equivalent classes • P31. Defining wrong equivalent classes • P32. Several classes with the same label • P33. Creating a property chain with just one property
Logical consistency	
• P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P14. Misusing "owl:allValuesFrom" • P15. Misusing "not some" and "some not" • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships • P31. Defining wrong equivalent classes • P33. Creating a property chain with just one property	
Real world representation	
• P5. Defining wrong inverse relationships • P9. Missing basic information • P10. Missing disjointness • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships	**Ontology language specification**
	• P34. Untyped class • P35. Untyped property

Abbildung 97: *Pitfall* Katalog nach María Poveda [137]

Critical (1)

P01. Creating polysemous elements
P03. Creating the relationship "is" instead of using "rdfs:subClassOf", "rdf:type" or "owl:sameAs"
P05. Defining wrong inverse relationships
P06. Including cycles in a class hierarchy
P14. Misusing "owl:allValuesFrom"
P15. Using "some not" in place of "not some"
P16. Using a primitive class in place of a defined one
P19. Defining multiple domains or ranges in properties
P27. Defining wrong equivalent properties
P28. Defining wrong symmetric relationships
P29. Defining wrong transitive relationships
P31. Defining wrong equivalent classes
P37. Ontology not available on the Web
P39. Ambiguous namespace
P40. Namespace hijacking

Important (2)

P10. Missing disjointness
P11. Missing domain or range in properties
P12. Equivalent properties not explicitly declared
P17. Overspecializing a hierarchy
P18. Overspecializing the domain or range
P23. Duplicating a datatype already provided by the implementation language
P24. Using recursive definitions
P25. Defining a relationship as inverse to itself
P26. Defining inverse relationships for a symmetric one
P30. Equivalent classes not explicitly declared
P34. Untyped class
P35. Untyped property
P38. No OWL ontology declaration
P41. No license declared

Minor (3)

P02. Creating synonyms as classes
P04. Creating unconnected ontology elements
P07. Merging different concepts in the same class
P08. Missing annotations
P09. Missing domain information
P13. Inverse relationships not explicitly declared
P20. Misusing ontology annotations
P21. Using a miscellaneous class
P22. Using different naming conventions in the ontology
P32. Several classes with the same label
P33. Creating a property chain with just one property
P36. URI contains file extension

Abbildung 98: Kategorisierung der Pitfalls nach Priorität

Nicht alle *Pitfalls* haben die gleiche Priorität. Während einige *Pitfalls* als kritisch betrachtet werden, sind andere wiederum nur Empfehlungen. Je nach Modellierungstiefe, Größe der Ontologie und Einsatz können *Pitfalls* ignoriert werden. Insgesamt wird in [81] zwischen folgenden Stufen unterschieden (siehe Abbildung 98 für die Kategorisierung aller *Pitfalls*):

- *Kritisch (Critical)*: Es ist wichtig, diese Fehler zu beheben. Andernfalls können sie Auswirkungen auf die Konsistenz, die Inferenzmechanismen und die Anwendbarkeit der Ontologie haben.
- *Wichtig (Important)*: Die Fehler sind nicht schwerwiegend für die Funktion und den Einsatz der Ontologie, es ist aber zu empfehlen, sie zu beheben.
- *Unbedeutend (Minor)*: Es handelt sich nicht wirklich um einen Fehler, aber das Beheben würde die Ontologie stimmiger machen.

Die vollständige Definition eines Pitfalls (P05) zeigt Abbildung 102. Sie beinhaltet Informationen über die Priorität (*Importance Level*), das betreffende Objekt (*Affects to*), eine Beschreibung (*Description*), ein Beispiel (*Examples*) und Vorschläge, wie dieses Problem behoben werden kann (*How to solve it*). Eine vollständige Definition aller *Pitfalls* kann [81] entnommen werden.

Weitere Beispiele für häufig vorkommende Designfehler sind folgende:

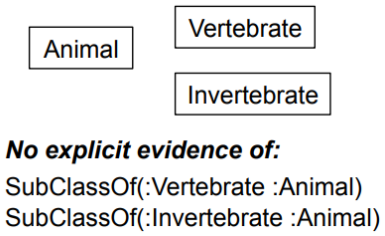


Abbildung 99: *Minor*: Erstellen von Klassen, die innerhalb der Ontologie nicht verknüpft sind (P04) [81]

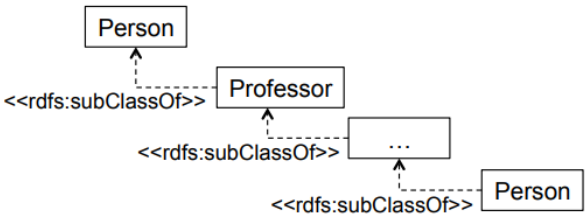


Abbildung 100: *Critical*: Erstellen von Zyklen innerhalb von Klassenhierarchien (P06) [81]

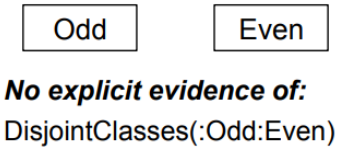


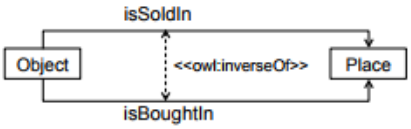
Abbildung 101: *Important*: Fehlende Deklaration von Disjunktheit (P10) [81]

Title	P05. Defining wrong inverse relationships	**Importance level**	Critical
Aspects	Wrong inference	**Affects to**	Object properties

Description

Two relationships are defined as inverse relations when they are not necessarily inverse.

Examples

Graphical representation and/or OWL code	Natural language description
	In the graphical example we can see the result of considering two converse* relationships, "isSoldIn" and "isBoughtIn", as inverse. This fact could cause an error at the semantic level, since if we have "object1 isSoldIn place3", then the reasoner will infer that "place3 isBoughtIn object1" which is semantically wrong. In addition, as the domain of "isBoughtIn" is set as "Object", the individual "place3" will be classified as "Object", which is also wrong. *Note: converse words are pairs of words that refer to a relationship from opposite points of view, such as buy/sell, parent/child, among others.

How to solve it

In order to solve this pitfall one should first check the following points:

(a) Check that whenever a relationship holds between individuals "a" and "b", the inverse relationship necessarily holds between "b" and "a" and that the relations have the same semantics.

(b) Check whether the expressions used to define the domain of one relationship denote the same set of individuals denoted by the range of the other property, even though they might involve different class restrictions. Making this verification easier is one of the reasons why we recommend defining the domain and range of the relationships whenever possible.

Once the checks are done, the following actions might be taken:

Case 1: When (a) holds but (b) does not. If the relationships are actually inverse but the domain of one of them does not match the range of the other relationship, then the domains and ranges should be modified. One should make sure that the expression used in the domain of one relationship represents the same set of individuals represented by the range of the other property even though they are not using exactly the same class.

Case 2: When (a) does not hold. If it is not clear that the relationships are actually inverse, then the `owl:inverseOf` axiom should be removed. For addition information about creating inverse properties see "P13. Inverse relationships not explicitly declared".

Abbildung 102: Pitfall P05: Definieren von falschen inversen Beziehungen [81]

Zur Detektion eventuell vorhandener Pitfalls wurde auf folgende Ansätze zurückgegriffen:

- *Structural pattern matching*: Die interne Struktur der Ontologie wird mit Blick auf bestimmte Modelle innerhalb der Ontologie untersucht. Wird ein bestimmtes Muster innerhalb von Modellen wiedererkannt, wird die zugehörige *Pitfall* aktiviert.
- *Linguistic analysis*: Die Ontologie wird nach inhaltsbezogenen Informationen, wie Annotationen oder URIs, durchsucht. Zusätzlich wird die lexikalische Datenbank WordNet[36] überprüft, um festzustellen, ob z.B. zwei als unterschiedlich markierte Konzepte synonym sind.

[36] https://wordnet.princeton.edu/

- *Specific characteristic search*: Sechs automatisierte Methoden prüfen die generellen Charakteristika einer Ontologie, welche weder „*Structural pattern matching*“ noch „*Linguistic analysis*“ zugeordnet werden können.

7.1.3 Ergebnisse

Im folgenden Kapitel werden die gewonnenen Erkenntnisse aus der strukturellen Bewertung und die daraus resultierenden Schlussfolgerungen zusammengefasst.

7.1.3.1 Kontinuierliche Verifikation mit Hermit

Bei dem Einsatz der Inferenzmaschine *Hermit* während des Entwicklungsprozesses der Ontologie handelte es sich weniger um ein konkretes Experiment, sondern um eine kontinuierliche Verifikation, aus der verschiedene Erkenntnisse gewonnen werden konnten.

Ist der zu modellierende Kontext klein und überschaubar, kann die Validität der Struktur ohne Werkzeugunterstützung gewährleistet werden. Steigen jedoch die Komplexität und Anzahl der Axiome, stellt man fest, dass es nicht möglich ist, alle potentiellen Fehlerquellen zu erkennen. Für die Entwicklung eines komplexen Ressourcenmodells, wie es in dieser Arbeit vorgestellt wurde, hilft eine Inferenzmaschine wie *Hermit*, konzeptionelle und logische Fehler zu identifizieren bzw. diese zu vermeiden.

Damit *Hermit* dazu in der Lage ist, muss der Entwickler die zur Verfügung stehenden Sprachkonstrukte zur Beschreibung von Beziehungen und Abhängigkeiten nutzen. Bei der Verifikation hat sich gezeigt, dass dabei ein strukturiertes Vorgehen beim Aufbau der Taxonomie (Stichpunkt Vererbung) die Wahl geeigneter *Objectproperties* und die Angabe von

1. Disjunktion bei Klassen und Properties und
2. Domain- und Range- Restriktionen bei Properties

besonders hilfreich war. Gerade bei komplexen Beziehungen und Vererbung konnte so schnell festgestellt werden, ob ein Fehler bei der (logischen) Modellierung der Ontologie gemacht wurde.

Für den Aufbau der Klassen- und Objectproperties-Hierarchie kann keine Empfehlung gegeben werden, da diese immer stark von der Domäne und dem Kontext der zu modellierenden Gegebenheit abhängig ist.

Bei der Angabe von Disjunktionen und Domain- bzw. Range- Restriktionen muss entschieden werden, wie sehr man den Benutzer bei der Verwendung der Ontologie einschränken möchte (überlässt man dem Benutzer die richtige Verwendung der modellierten Konzepte oder stellt man sicher, dass der Benutzer nicht in der Lage ist, Konzepte falsch zu benutzen). Mit jeder Restriktion steigt die Komplexität der Ontologie. Die Vor- und Nachteile einer möglichst vollständigen Beschreibung eines Modells durch z.B. Angabe von Vererbungshierarchien, Disjunktion und Restriktionen sind wie folgt zusammengefasst:

Pro:
+ Modellgenauigkeit
+ Es können weniger falsche Beziehungen beschrieben werden
+ Beschreibung eines Modells wird restriktiver
+ Semantik für Klassen bzw. Beziehungen wird für einen Anwender ersichtlicher

Contra:
- Komplexität steigt
- Höhere Anzahl von Axiomen
- Höhere Abfragezeiten von Informationen
- Aufwand bei neuen Inhalten bzw. Erweiterung einer Ontologie steigt
- Es bedarf mehr verschiedener Beziehungen, um Konzepte zu beschreiben

Tabelle 19 zeigt die Auswertung der Axiome für das Beispiel aus Abbildung 92.

	Axiome ohne Reasoning		
	Klassen	Vererbungshierarchien	Disjunktion
mit Disjunktion	25	24	**8**
ohne Disjunktion	25	24	**0**

	Axiome mit Reasoning		
	Klassen	Vererbungshierarchien	Disjunktion
mit Disjunktion	25	24	**216**
ohne Disjunktion	25	24	**0**

Tabelle 19: Komplexitätszuwachs durch Angabe von Disjunktion

Das Beispiel aus Abbildung 92 wird in Form von 57 Axiomen (inferiert 265 Axiome) beschrieben:

- 25 Axiome für Klassen,
- 24 Klassen für Vererbungshierarchien,
- 8 Axiome für Disjunktion.

Ohne die Disjunktionen kann das Beispiel mit 49 Axiomen (inferiert ebenfalls 49 Axiome) beschrieben werden. Während der Komplexitätsunterschied ohne Inferenz gering ist (8 Axiome weniger), ist die Auswirkung nach der Inferenz deutlich. Da aufgrund der Vererbung für jede Klasse die Disjunktion beschrieben werden muss, werden insgesamt 216 neue Axiome hinzugefügt.

Die Auswertung des Anstiegs der Axiome je nach Befüllungsgrad der Ontologie ist in Diagramm 1 zusehen. Für diese Auswertung wurden die importierten Ontologien ignoriert, sodass nur die Auswirkungen innerhalb der zu entwickelnden Ontologie zu sehen sind.

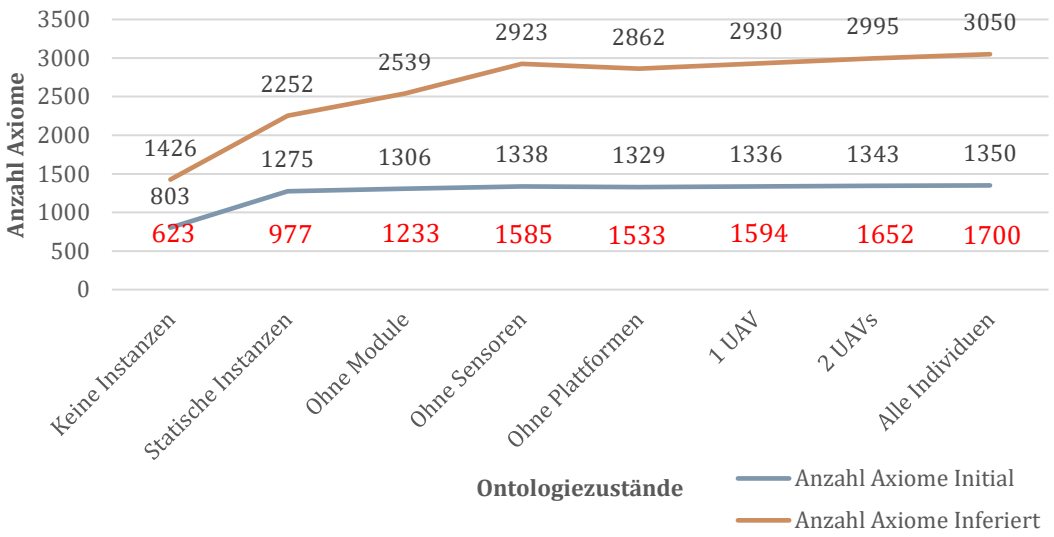


Diagramm 1: Anzahl der Axiome innerhalb der Ontologie zu verschiedenen Zuständen

Für die Auswertung wurden acht repräsentative Zustände bestimmt, deren Wahl im Folgenden erläutert wird:

- *Keine Instanzen*: Dies ist der Zustand der Ontologie, nachdem der Designprozess abgeschlossen war. Die Klassen und Beziehungshierarchie standen fest. Für den produktiven Einsatz musste die Ontologie nur noch mit konkreten Instanzen befüllt werden.
- *Statische Instanzen*: Die Ontologie wurde mit Instanzen von Klassen wie z.B. Fähigkeiten, Umweltzuständen und Entitäten befüllt, die zur weiteren Beschreibung von Zusammenhängen zwischen z.B. Sensoren und Plattformen notwendig waren.
- *Ohne Module*: Die Ontologie ist bis auf die Module vollständig, so wie in Kapitel 6.2 beschrieben, befüllt. In diesem Zustand können keine perzeptiven Aufgaben durchgeführt werden.
- *Ohne Sensoren*: Die Ontologie ist bis auf die Sensoren vollständig, so wie in Kapitel 6.2 beschrieben. Es können keine Bilddaten zur Verfügung gestellt werden.
- *Ohne Plattformen*: Die Ontologie ist bis auf die Plattformen vollständig, so wie in Kapitel 6.2 beschrieben, befüllt.
- *1 UAV*: Die Ontologie ist vollständig, so wie in Kapitel 6.2 beschrieben, befüllt. Alle Plattformen, bis auf eine, wurden entfernt. Dies ist der Zustand einer Ontologie, wie sie sich an Bord eines UAVs während einer Mission befinden würde.
- *2 UAVS*: Der Ontologiezustand ist mit dem der 1 UAV-Konfiguration identisch, nur dass in diesem Fall 2 UAVs existieren. Diese Konfiguration ist für einen Multi-UAV-Einsatz gedacht.
- *Alle Individuen*: Die Ontologie ist vollständig, so wie in Kapitel 6.2 beschrieben, befüllt. Diese Ontologie wird verwendet, um vor der Missionsdurchführung eine UAV-Konfiguration zu bestimmen.

Die Anzahl der Axiome ist sowohl zum initialen Zustand (blaue Linie) als auch zum inferierten Zustand (orangene Linie) abgebildet. Zusätzlich ist in Rot die Differenz der Anzahl der Axiome zu den verschiedenen Zuständen zu sehen. Der höchste Anstieg der Axiome fand statt, sobald die Ontologie mit Instanzen befüllt wurde. Dies lag daran, dass für jede Instanz, je nach Komplexität, eine große Anzahl an Axiomen geschrieben werden musste.

Der Anstieg der Axiome zeigt, dass sich die Größe und Komplexität einer Ontologie stark verändert, wenn Sprachelemente wie Disjunktion oder Restriktionen verwendet werden. Die Differenz zwischen dem Zustand „Keine Instanzen“ und dem Zustand „Alle Individuen“ beträgt

- 75, wenn auf kein implizites Wissen mittels einer Inferenzmaschine geschlossen wird,
- 798, wenn auf implizites Wissen geschlossen wird.

Dies setzt sich aus den modellierten Klassen und deren Beziehungen zusammen. Für jede Instanz wird, je nach Komplexität der modellierten Klasse, eine unterschiedliche Anzahl von Axiomen inferiert. Abbildung 103 zeigt beispielhaft die Beschreibung der Klasse *OctoXL* und der Klasse *SonyFCB*. Es wird deutlich, dass die Anzahl der Eigenschaften und damit die Anzahl der Axiome je nach Klasse stark variieren. Während die linke Klasse 17 Axiome zur Beschreibung benutzt, hat die rechte Klasse lediglich 13 beschreibende Axiome.

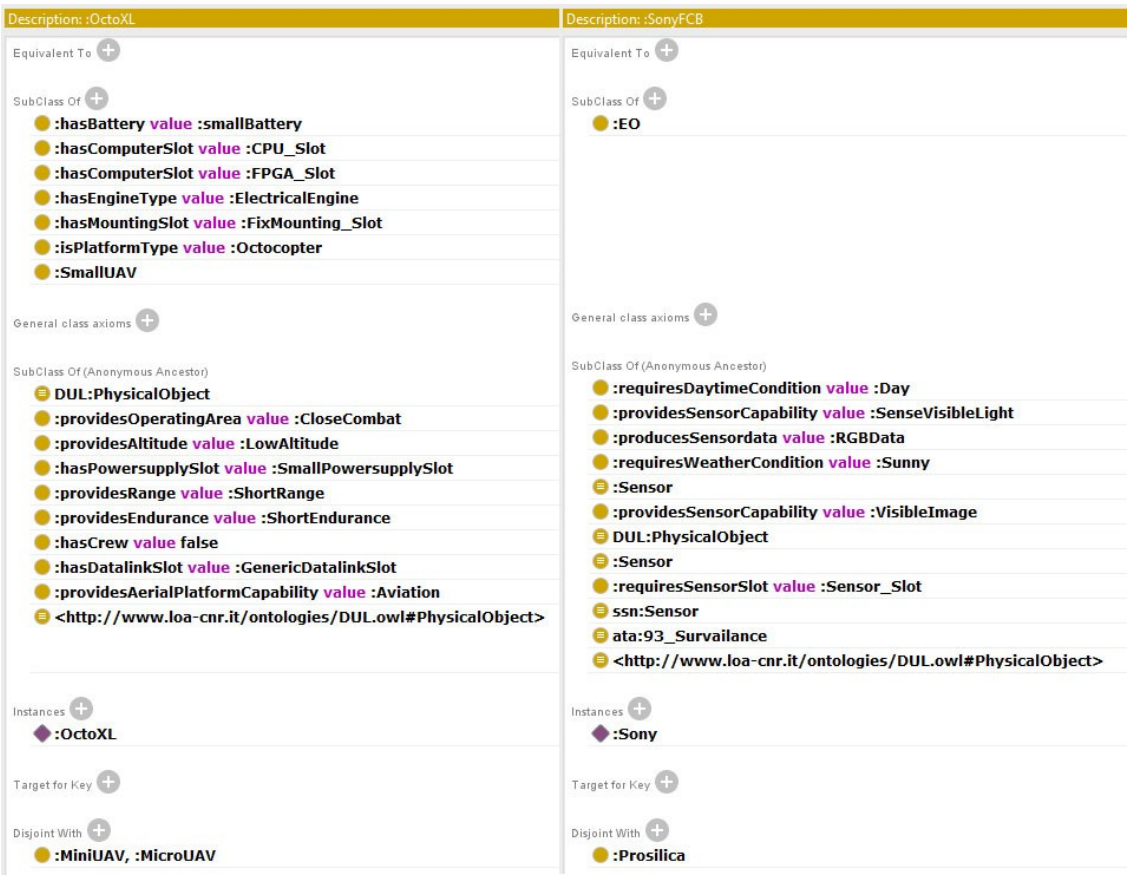


Abbildung 103: Vergleich der Modellierung zweier Klassen. Links die Beschreibung für die Klasse OctoXL, rechts die Beschreibung für die Klasse SonyFCB

Wird z.B. eine Instanz der Klasse *SonyFCB* zur Ontologie hinzugefügt, benötigt man ein Axiom, um eine Aussage darüber zu machen, dass die Instanz *Sony* der Klasse *SonyFCB* angehört (vgl. Abbildung 104).

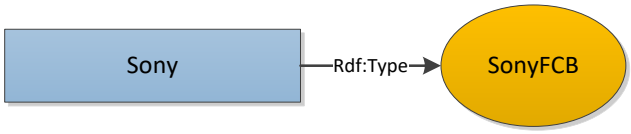


Abbildung 104: "Sony" Instanz der Klasse SonyFCB

Zusätzlich werden weitere Axiome dem Individuum *Sony* zugeteilt, wodurch sich die Gesamtzahl der Axiome während des Inferenzvorgangs erhöht. Abbildung 105 zeigt beispielhaft eine vollständige Beschreibung des *Sony* Individuums, nachdem alle Eigenschaften durch die Inferenzmaschine zugeordnet wurden. Insgesamt besteht die Beschreibung aus 20 Axiomen.

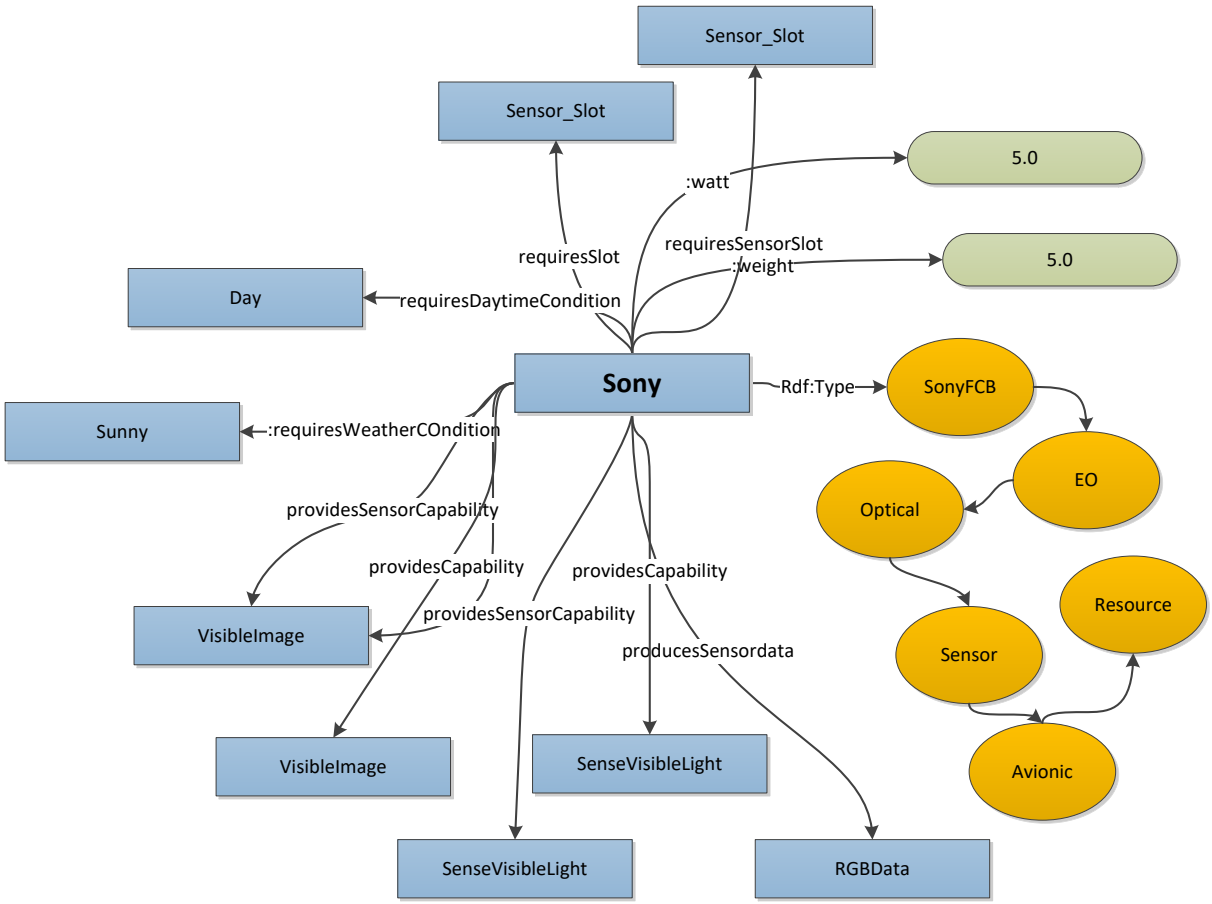


Abbildung 105: Vollständige Beschreibung des Individuums "Sony", nachdem die Inferenzmaschine weitere Eigenschaften zugeordnet hat

Blaue Rechtecke repräsentieren andere Instanzen. In grün sind die *Dataproperties* dargestellt. Gelbe Ellipsen repräsentieren die Klassen, zu denen der *Sony* Sensor gehört. Die Pfeile bezeichnen die Beziehungen, mit denen die einzelnen Instanzen verbunden sind.

Die Inferenzmaschine Hermit wurde genutzt, um sicherzustellen, dass die Ontologie konsistent ist, d.h. keine Widersprüche aufweist. Weiterhin wurde durch Hermit auf implizites Wissen

geschlossen. Durch Angabe von Disjunktionen und Range- bzw. Domain- Restriktionen wurde zwar die Komplexität erhöht, gleichzeitig aber sichergestellt, dass Instanzen und Properties nicht in einem falschen Kontext benutzt werden konnten. Dabei wurde darauf geachtet, dass die Restriktionen hinreichend genug waren, um Aussagen logisch korrekt zu beschreiben, die Komplexität jedoch im Rahmen bleibt, sodass die Ontologie zur Beschreibung des zu entwickelnden Ressourcenmodells an Bord eines UAVs genutzt werden kann. Schlussfolgernd ist festzuhalten, dass beim Entwurf der Ontologie entschieden werden muss, wo die Priorität liegt:

- Die Angabe vieler Restriktionen steigert die Komplexität der Ontologie, stellt aber sicher, dass (1) keine unlogischen Zusammenhänge modelliert werden und dass (2) Konzepte und Beziehungen von einem Benutzer nicht in einem falschen Kontext verwendet werden.
- Der Verzicht von Beschränkungen senkt die Komplexität und Zugriffszeiten beim Schreiben bzw. Lesen der Ontologie, jedoch wird dem Benutzer die Aufgabe überlassen, die Ontologie korrekt zu verwenden. In diesem Fall musst der Benutzer sicherstellen, dass Beziehungen zwischen Instanzen korrekt angegeben sind.

7.1.3.2 Abschlussverifikation mit dem Pitfall Scanner

Am Ende des Designprozesses wurde der Pitfall Scanner dazu verwendet, um zu überprüfen, ob für die Verwendung der Ontologie im Rahmen dieser Arbeit kritische Fehler in der Ontologie enthalten sind und falls ja, wie sie beseitigt werden müssen. Die Auswertung des Scanners für die Ontologie (vgl. Kapitel 6.2) ist in Abbildung 106 zu sehen.

Results for P04: Creating unconnected ontology elements.	8 cases | Minor
Results for P08: Missing annotations.	339 cases | Minor
Results for P11: Missing domain or range in properties.	128 cases | Important
Results for P13: Inverse relationships not explicitly declared.	25 cases | Minor
Results for P22: Using different naming conventions in the ontology.	ontology* | Minor
Results for P30: Equivalent classes not explicitly declared.	1 case | Important
Results for P41: No license declared.	ontology* | Important

Abbildung 106: Ergebnis des Scanners für die zu entwickelnde Ontologie

Wie bereits in Kapitel 7.1.2.2 erwähnt, gibt der Scanner darüber Aufschluss, welche *Pitfalls* in der Ontologie enthalten und wie kritisch diese sind. Ob sie letzten Endes beseitigt werden sollen bzw. müssen, obliegt der Entscheidung des Designers und dem zugrunde liegenden Anwendungsfall. Im Rahmen des Evaluierungsprozesses wurden verschiedene *Pitfalls* identifiziert. Auf das Beheben dieser wurde jedoch bewusst verzichtet. Im Folgenden werden die identifizierten *Pitfalls* aufgezählt und erläutert, warum sie im Rahmen dieser Arbeit nicht beseitigt wurden:

- P 04: Der Scanner merkt an, dass Konzepte gefunden wurden, die nicht verwendet werden. Es gibt Konzepte, die zu Interoperabilitätszwecken in der Ontologie aufgenommen, jedoch zum Zeitpunkt dieser Arbeit noch nicht genutzt wurden. Dies könnte wichtig sein, falls die Ontologie im Rahmen einer Erweiterung benutzt wird.
- P 08: Es fehlen Annotationen wie z.B. Informationen über den Autor oder textuelle Beschreibungen von Klassen. Auf Annotationen (Kommentare) für Klassen wurde im Rahmen dieser *Proof of Concept* Implementierung verzichtet.
- P 11: Es fehlen Domain- und Range- Restriktionen. Wie erwähnt, wurde bewusst auf einige Restriktionen verzichtet, damit die Komplexität und Größe der Ontologie an Bord des UAVs verarbeitbar bleibt.
- P 13: Nicht für alle Beziehungen existieren ihre inversen Beziehungen. Es war nicht notwendig, für alle Beziehungen eine inverse Beziehung zu deklarieren. Um die Komplexität und Größe der Ontologie gering zu halten, wurde nicht für jede Beziehung ihre Inverse angelegt.
- P 22: Die Namenskonventionen sind nicht durchgehend gleich (z.B. Uppercase und Camelcase). Durch Integration von Ontologien Dritter kommt es zu verschiedenen Namenskonventionen.
- P 30: Es existiert eine äquivalente Klasse, die nicht als solche deklariert wurde. Der *Pitfall* Scanner geht davon aus, dass die Konzepte *Detection* und *Sensing* synonym sind, weshalb diese als gleichwertig (*owl*:*equivalentClass*) detektiert werden sollten. In diesem konkreten Fall haben die zwei Begriffe verschiedene Bedeutungen, wodurch die beiden Konzepte korrekterweise nicht als gleichwertig deklariert wurden (vgl. Abbildung 96).
- P 41: Es fehlt eine Lizenz, um zu bestimmen, wie diese Ontologie verwendet werden darf. Für diese Ontologie wurde keine Lizenz bestimmt, da es sich um eine wissenschaftliche Arbeit handelt, bei der ein *Proof of Concept* erarbeitet wurde.

Wie bereits erwähnt, können mittels des Scanners Fehler identifiziert werden, die durch eine Inferenzmaschine nicht als solche identifiziert werden. Durch den Einsatz einer Inferenzmaschine zur Validierung der Konsistenz der Ontologie wurden kritische *Pitfalls* vermieden. Der Scanner machte jedoch darauf aufmerksam, dass nach wie vor weniger kritische Fehler in der Ontologie enthalten sind. Obwohl die erkannten Fehler bewusst in Kauf genommen wurden, sollten die Pitfalls für die zukünftige Erweiterung der Ontologie im Auge behalten werden.

Möchte man eine strukturelle Bewertung einer Ontologie durchführen, kann dies mittels einer Inferenzmaschine erfolgen. Zusätzlich ist der Pitfall Scanner eine gute Ergänzung, um Fehler zu detektieren, die von einer Inferenzmaschine nicht erkannt werden können.

7.1.4 Fazit der strukturellen Bewertung

Im Rahmen der strukturellen Bewertung wurde untersucht, wie und welche allgemeingültigen domänenunabhängigen Designkriterien eingehalten werden müssen, um eine konsistente On-

tologie zu modellieren und welche Vor- bzw. Nachteile dies mit sich bringt. Für die Experimente kamen zwei Methoden zum Einsatz: Das Überprüfen inkonsistenter Beschreibungen mittels der Inferenzmaschine *Hermit* und das Überprüfen von empfohlenen Designregeln und *best practices* mittels des *Pitfall* Scanners.

Der kontinuierliche Einsatz einer Inferenzmaschine während des Designs der Ontologie zeigte, dass bereits während der Modellierungsphase Fehler, wie z.B. inkonsistente Beschreibungen von Klassen, identifiziert und behoben werden können. Dies ist besonders dann relevant, wenn die Ontologie eine gewisse Größe und Komplexität übersteigt. Durch die Fähigkeit der Inferenzmaschine, die Ontologie auf Regeln wie Disjunktheit oder Domain- und Range- Restriktionen hin zu prüfen, wird der Designer in die Lage versetzt, eine Ontologie so zu modellieren, dass eine falsche Benutzung durch den Anwender erschwert wird. Wie sich zeigt, steigen jedoch dadurch (siehe Kapitel 7.1.3, Tabelle 19) die Komplexität der Ontologie und die Anzahl der Axiome. Dies wiederum wirkt sich auf die Inferenz- und Abfragedauer der Ontologie aus. Entscheidet sich der Designer für eine losere Beschreibung der Konzepte, wird die Komplexität verringert und die Abfragedauer verkürzt. Letztlich liegt es in der Verantwortung des Benutzers, die Konzepte der Ontologie korrekt einzusetzen.

Um am Ende des Designprozesses festzustellen, ob *best practices* eingehalten wurden und um Eigenschaften zu validieren, die mit *Hermit* nicht untersucht werden konnten, wurde das Werkzeug *OntOlogy Pitfall Scanner (OOPS!)* [81] eingesetzt. Der Scanner zeigt eventuell vorhandene Fehler auf, die sich wiederum durch verschiedene Stufen (*critical*, *important* und *minor*) unterscheiden. Je nach Fehlerart werden unterschiedliche Korrekturempfehlungen (z.B. das Ergänzen von Restriktionen oder Hinzufügen bestimmter Elemente) angeboten. Es obliegt dann dem Designer, diese umzusetzen. Schließlich kann das Ergebnis des Scanners dafür genutzt werden, einem Kunden oder Benutzer zu garantieren, dass die modellierte Ontologie allgemeingültigen Designkriterien und *best practices* erfüllt, keine schwerwiegenden Fehler aufweist und damit eine gewisse Qualität der Ontologie gewährleistet werden kann.

7.2 Operationelle Bewertung

Aus Kapitel 1.4 ist bekannt, dass die Ontologie zwei Anwendungsfälle bedient:

1. Zur Zeit der Missionsvorbereitung soll festgestellt werden, welche Ressourcen insgesamt zur Verfügung stehen und wie ein UAV konfiguriert sein muss, um eine spezifische Mission durchführen zu können.
2. Während der Missionsdurchführung soll aufgabenabhängig und situationssensitiv identifiziert werden, ob und welche Ressourcen für eine Aufklärungsaufgabe bereitstehen und zu verwenden sind.

In der operationellen Validierung wurde untersucht, wie die Ontologie bzw. das S&PM für beide Anwendungsfälle eingesetzt werden kann und dabei den Anforderungen aus Kapitel 3 gerecht wird.

7.2.1 Missionsvorbereitung

Zunächst soll in Experimenten untersucht werden, wie und ob die Ontologie für die Missionsvorbereitung eingesetzt werden kann.

7.2.1.1 Untersuchungsgegenstand

Hierbei wurde untersucht, wie die Ontologie angewendet werden kann, um aus einem Bestand von Ressourcen, bestehend aus mehreren UAV-Plattformen, Sensoren, Rechnersystemen und Softwarealgorithmen (vgl. Kapitel 6.2), eine möglichst optimale Konfiguration zu finden. Dabei wurde nicht nur untersucht, welche Ressourcen vorhanden sind, sondern auch, ob und welche Ressourcen miteinander kombiniert werden konnten, um so eine missionstaugliche UAV-Konfiguration zu finden. Zur Ermittlung einer passenden Konfiguration wurden zwei Herangehensweisen untersucht.

1. Durch Angabe missionsspezifischer Parameter sollte eine bestimmte UAV-Konfiguration vorgeschlagen werden. (*Top-Down*)
2. Durch Definition ausgewählter Ressourcen sollten die daraus resultierenden Aufklärungsfähigkeiten ermittelt werden. Anhand der angezeigten Fähigkeiten konnte anschließend entschieden werden, welche Konfiguration von Interesse ist. (*Bottom-Up*)

Abhängig von Größe und Leistungsvermögen kann ein UAV nur mit bestimmten Sensoren und Rechnersystemen ausgestattet werden. In den meisten Fällen existieren UAVs heutzutage mit einer ganz bestimmten Konfiguration, die für spezifische Aufgaben konzipiert wurden. Damit dieses Konzept umgesetzt werden kann, wurde von einer (zukünftigen) *Plug & Play* Fähigkeit sowie einem modularen Aufbau der Ressourcen ausgegangen.

Top-Down Ansatz

Zur Ermittlung einer UAV-Konfiguration für eine definierte Missionsaufgabe konnten die folgenden missionsspezifischen Parameter (*freie Parameter*) angegeben werden (vgl. Tabellen aus Kapitel 6.2.5):

- Methode, z.B. detektieren,
- Entität (was soll detektiert werden?), z.B. Fahrzeug,
- Missionsdauer, z.B. 2 Stunden,
- Missionszeitraum, z.B. 23:30 Uhr – 01:30 Uhr,
- Beschränkung, z.B. nur Fahrzeuge auf Straßen.

Von der Wahl der ersten beiden Parameter hängt z.B. die Wahl der Bildverarbeitungsmodule ab. Über die Missionsdauer kann ermittelt werden, welche UAVs aufgrund von Reichweitenbegrenzung nicht herangezogen werden können. Der Missionszeitraum hat z.B. Einfluss auf die Sensorwahl (IR vs. EO). Bei Beschränkungen können weitere Kriterien angegeben werden, die nicht über die freien Parameter festgelegt werden.

Bottom-Up Ansatz

Liegt das Interesse nicht auf UAV-Konfigurationen für eine vorgegebene Mission, sondern auf der Auskunft vorhandener Fähigkeiten, die durch eine Auswahl von Ressourcen bereitgestellt werden können, konnten die folgenden freien Parameter angegeben werden:

- Plattform, z.B. Sagitta,
- Sensorhalterung, z.B. DST-Gimbal,
- Sensor, z.B. Sony-FCB,
- Hardware, z.B. Intel-i7.

7.2.1.2 Methode und Experimente

Die Ontologie wurde zur Missionsvorbereitung auf zwei Arten verwendet. Im Top-Down Ansatz wurde untersucht, wie man von einer Missionskonfiguration zu einer UAV-Konfiguration kommt und im Bottom-Up Ansatz, welche Aufgaben mit einer fest vorgegebenen UAV-Konfiguration durchgeführt werden konnten.

Top-Down Ansatz: Zur Ermittlung einer UAV-Konfiguration anhand der in Kapitel 7.2.1.1 vorgegebenen freien Parameter wurden drei Missionen spezifiziert (siehe Tabelle 20).

Mission 1	
Methode	Detektion
Entität	Fahrzeug
Dauer	2 Stunden
Zeit	10:00 – 12:00
Beschr.	-

Mission 2	
Methode	Streamen
Entität	-
Dauer	24 Stunden
Zeit	-
Beschr.	Farbbild

Mission 3	
Methode	Klassifizieren
Entität	Zivilist
Dauer	1 Stunde
Zeit	22:00-23:00
Beschr.	-

Tabelle 20: Anforderungen zur Ermittlung geeignet UAV-Konfigurationen

In der ersten Mission sollte eine Fahrzeugdetektion um die Mittagszeit durchgeführt werden. In der zweiten Mission sollten Bilddaten übertragen werden. Die Missionsdauer betrug 24 Stunden und die Bilddaten sollten RGB-kodiert sein. Bei der dritten Mission sollten in einem Zeitraum von einer Stunde Erkennungen als Zivilisten klassifiziert werden.

Bottom-Up Ansatz: Um feststellen zu können, welche Aufgaben mittels einer ganz bestimmten UAV-Konfiguration durchgeführt werden können, wurde die semantische Beschreibung der einzelnen Komponenten einer UAV-Konfiguration untersucht. In Abbildung 107 sind beispielhaft die Beschreibungen für eine Plattform, einen Sensor und ein Bildverarbeitungsmodul zusehen.

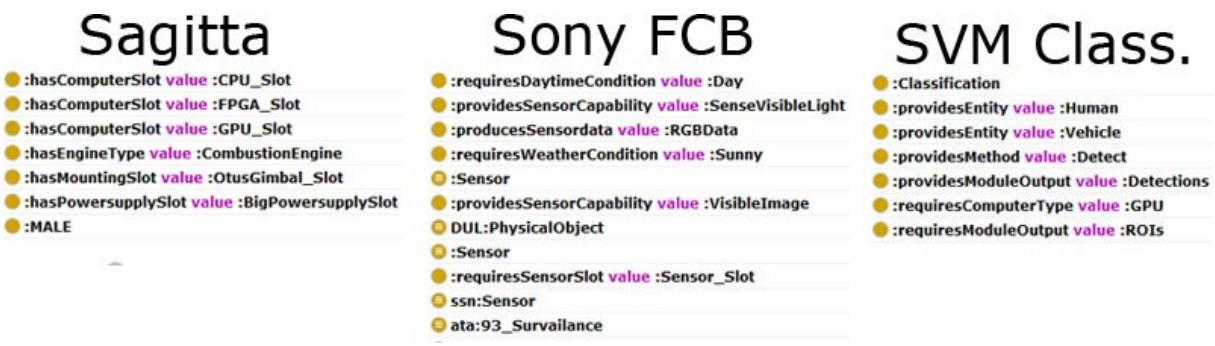


Abbildung 107: Semantische Beschreibung einer Plattform, eines Sensors und eines Bildverarbeitungsmoduls

Die Abbildung zeigt, dass das Modul „SVM Class“ die Methode *Detect* und die Entitäten *Human* und *Vehicle* bereitstellt. Mit diesem Modul können sowohl Fahrzeuge als auch Personen detektiert werden. Das Modul muss auf einer GPU ausgeführt werden.

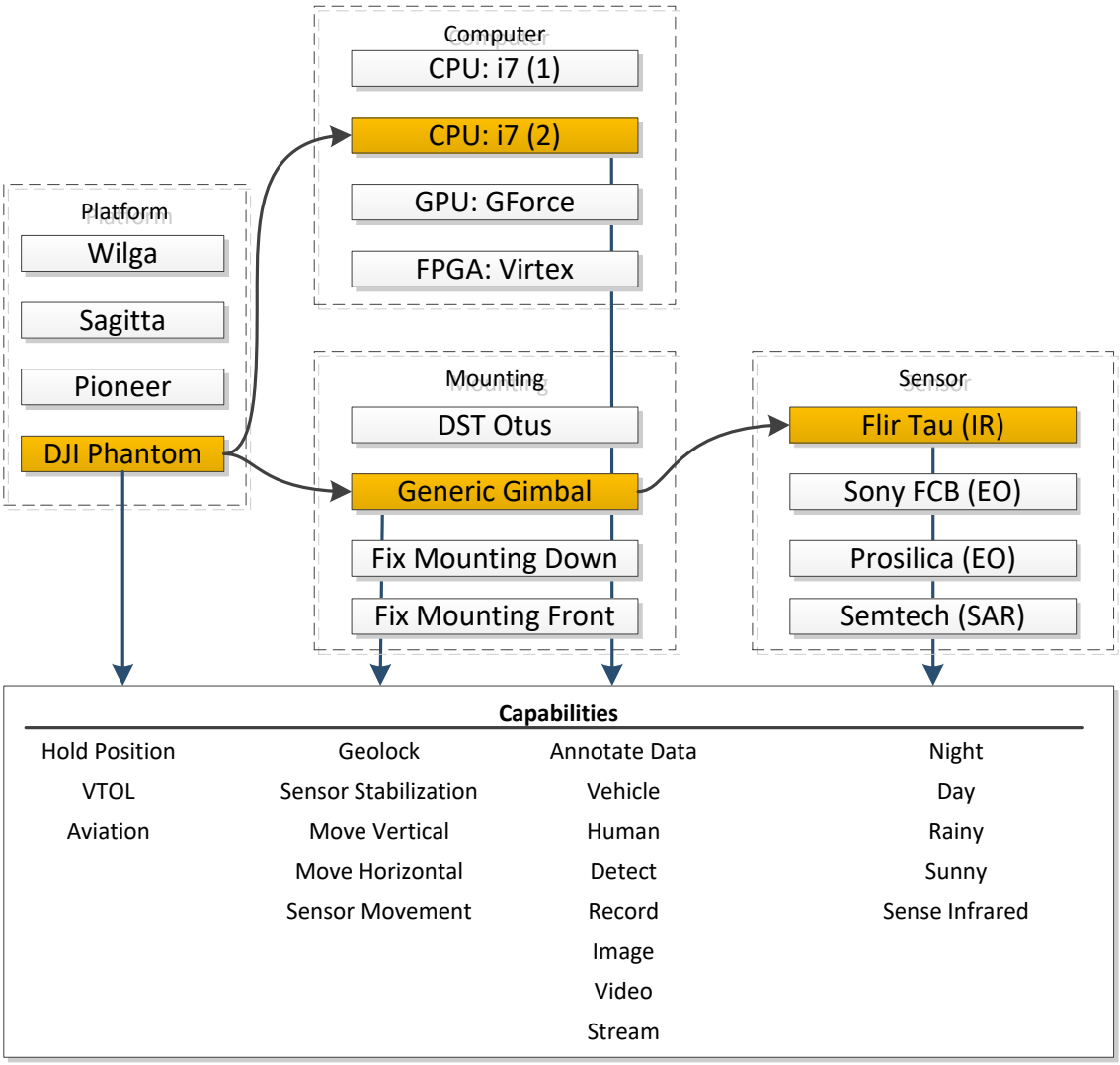


Abbildung 108 Beispielkonfiguration einer Plattform

Für das Experiment galt die Annahme, dass drei vorkonfigurierte UAVs zur Verfügung stehen (vgl. Tabelle 21).

Konfiguration 1	
Plattform	DJI Phantom
Sensor	FlirTau (IR)
Mounting	DSTSingle
Computer	CPU

Konfiguration 2	
Plattform	Hornet-Nano
Sensor	FixCam (EO)
Mounting	Fix
Computer	-

Konfiguration 3	
Plattform	Sagitta-1
Sensor	SonyFCB (EO)
Mounting	DST Otus
Computer	CPU, GPU

Tabelle 21: Drei verschiedene UAV-Konfigurationen

Aus der Konfiguration konnte mittels der Ontologie ermittelt werden, welche Fähigkeiten von den einzelnen Konfigurationen bereitgestellt werden. Hierfür wurden die verschiedenen Komponenten der einzelnen Konfigurationen untersucht und die jeweiligen Fähigkeiten und Eigenschaften identifiziert. Abbildung 108 zeigt diesen Vorgang exemplarisch für Konfiguration 1.

7.2.1.3 Ergebnisse

Die Experimente demonstrierten, wie das Softwaresystem benutzt wurde, um im Top-Down-Ansatz UAV-Konfigurationen für vorgegebene Missionen zu identifizieren. Im Bottom-Up-Ansatz wurde anschließend, ausgehend von verschiedenen, fest definierten UAV-Konfigurationen, untersucht, welche Missionen mit den Konfigurationen durchgeführt werden konnten.

Top-Down Ansatz: Die in Kapitel 7.2.1.2 definierten Missionen führten in den Experimenten zu den folgenden Ergebnissen. Anhand der gewählten Parameter wurden vom Softwaresystem verschiedene Konfigurationen ermittelt[37]:

Missionsaufgabe 1

- Aufgrund der Missionsdauer kamen nur die folgenden UAVs in Betracht: *Hunter-H1, Pioneer-P1, Reaper-X2, Predator-P1, Eagle-E1* und *GlobalHawk-GER1*. Andere UAVs wurden ausgeschlossen, da deren Einsatzdauer weniger als zwei Stunden beträgt.
- Aus der Angabe der Missionszeit wurde geschlossen, dass alle Sensortypen in Frage kommen. Das heißt, *SonyFCB6300, ProsilicaGen, FlirTau* und *SarSensor*.
- Durch die Kombination „Detektion“ und „Fahrzeug“ wurden die folgenden Bildverarbeitungsmodule identifiziert: *Haar Classifier, IR Blob Detector, DPM Classifier.*
- Aus den Bildverarbeitungsmodulen konnte auf die benötigte Hardware geschlossen werden: *GForce_1* (GPU)*, I7_1(*CPU*)* und *Virtex_1(*FPGA*).*

In Tabelle 22 sind die möglichen Plattformen mit der konfigurierbaren Hardware (Sensoren und Computerausstattung) dargestellt.

[37] Für die Plattformen wurden, soweit vorhanden, reale Daten verwendet. Jedoch wurden an manchen Eigenschaften (z.B. Sensoren und Computerausstattung) im Rahmen dieser Arbeit Modifikationen vorgenommen, damit im Rahmen des Proof of Concept bestimmtes Systemverhalten demonstriert und validiert werden konnte.

GlobalHawk-Ger1	Eagle-E1	Predator-P1	Reaper-X2	Hunter-H1	Pioneer-P1
CPU/GPU/FPGA	CPU/GPU	CPU/GPU	GPU/FPGA	CPU	GPU
EO/SAR/IR	SAR/IR/EO	SAR/IR	EO/IR/SAR	EO	EO/EO

Tabelle 22: Mögliche Plattformen für Mission 1

Durch Inferenz wurde korrekt entschieden, dass der SAR-Sensor nicht in Frage kommt, da es keine Bildverarbeitungsmodule gibt, die mit seinen Daten Fahrzeuge detektieren können. Da der *Hunter-H1* lediglich mit einem *EO*-Sensor und einer *CPU* ausgestattet werden kann, der *IR Blob Detector* jedoch der einzige Algorithmus ist, der auf einer CPU ausgeführt wird, gab es keine gültige Konfiguration für die Plattform *Hunter-H1*. Auch die Plattform *Reaper-X2* wurde folgerichtig ausgeschlossen, da für die Vorverarbeitung nur Bildverarbeitungsmodule vorhanden sind, die auf einer CPU ausgeführt werden müssen. Selbiges galt für *Pioneer-P1*. Letztlich blieben die folgenden Konfigurationen übrig:

Plattform	Sensor	BV-Modul	Hardware
GlobalHawk-Ger1	SonyFCB6300|ProsilicaGen	DPM Classifier	I7_1, GForce_1
GlobalHawk-Ger1	SonyFCB6300|ProsilicaGen	Haar Classifier	I7_1, Virtex_1
GlobalHawk-Ger1	FlirTau	IR Blob Detector	I7_1
Eagle-E1	SonyFCB6300|ProsilicaGen	DPM Classifier	I7_1, GForce_1
Eagle-E1	FlirTau	IR Blob Detector	I7_1, GForce_1
Predator-P1	FlirTau	IR Blob Detector	I7_1

Tabelle 23: UAV-Konfigurationen für Mission 1

Das Experiment wurde an dieser Stelle beendet, jedoch könnte daraus in einem nächsten Schritt anhand verschiedener Metriken (z.B. Kosten) eine Konfiguration ausgewählt werden.

Missionsaufgabe 2

- Aufgrund einer Einsatzdauer von 24 Stunden kamen nur die Plattformen *GlobalHawk-Ger1* und *Eagle-E1* in Betracht.
- Die Missionsaufgabe spezifizierte keine Missionszeit. Jedoch wurde definiert, dass es sich bei den Sensordaten um Farbdaten handeln sollte. Somit kamen nur EO-Sensoren in Frage: *SonyFCB6300* und *ProsilicaGen*.
- Aus Angabe der Methode und der Tatsache, dass ein Farbbild erwartet wurde, wurde ermittelt, dass das BV-Modul *RGB Streaming* benötigt wird.
- Aus dem BV-Modul wurde geschlossen, dass als Hardware nur eine CPU in Frage kam: *i7_1*.

Aus diesen Daten konnten die folgenden Konfigurationen ermittelt werden:

Plattform	Sensor	BV-Modul	Hardware
GlobalHawk-Ger1	SonyFCB6300|ProsilicaGen	RGB Streaming	I7_1
Eagle-E1	SonyFCB6300|ProsilicaGen	RGB Streaming	I7_1

Tabelle 24: UAV-Konfigurationen für Mission 2

Missionsaufgabe 3

- Da die Einsatzdauer eine Stunde betragen sollte, wurden folgende Plattformen ermittelt: *GlobalHawk-Ger1, Eagle-E1, Predator-P1, Reaper-X2, Hunter-H1, Pioneer-P1.*
- Aufgrund der Uhrzeit wurde die Annahme gemacht, dass die Mission nachts stattfinden sollte. Daraus ergab sich die Schlussfolgerung, dass nur ein Infrarotsensor in Frage kommt: *FlirTau.*
- Durch die Kombination aus der Methode *Klassifizierung* und *Zivilist* als Entität konnten keine geeigneten Bildverarbeitungsmodule identifiziert werden.

Da keine gültigen Module identifiziert wurden, konnten korrekterweise für diese Mission keine validen UAV-Konfigurationen ermittelt werden.

Bottom-Up Ansatz:

Aus den in Kapitel 7.2.1.2 definierten UAV-Konfigurationen wurden für die einzelnen Konfigurationen folgende Fähigkeiten und Eigenschaften ermittelt:

Konfiguration 1:

- Hinsichtlich der Plattform wurde geschlossen, dass die Fähigkeiten *VTOL* und *Hold Position* verfügbar sind. Außerdem konnte aus der Plattformbeschreibung eine *Missionsdauer von 15 Minuten* abgeleitet werden.
- Aus dem Mounting konnte geschlossen werden, dass der Sensor *stabilisiert* ist und *bewegt* werden kann.
- Aus der Kombination der Computerhardware und dem Sensor ergab sich, dass *Fahrzeuge* und *Personen* sowohl bei *Nacht* als auch bei *Tag* detektiert werden können. Des Weiteren kann die Detektion ebenfalls bei den Wetterverhältnissen *Rainy* und *Sunny* durchgeführt werden. Neben der Detektion können *Bild-* und *Videodaten* sowohl *aufgezeichnet* als auch *gestreamt* werden.

Konfiguration 2:

- Aus der Plattformbeschreibung konnte ermittelt werden, dass die Fähigkeiten *VTOL* und *Hold Position* zur Verfügung stehen. Außerdem kann aus der Plattformbeschreibung eine *Missionsdauer von zwei Minuten* geschlossen werden.
- Aus dem Mounting wurde geschlossen, dass der Sensor *nicht bewegt* werden kann und in *Flugrichtung orientiert* ist.
- Aus den Angaben der Computerhardware und des Sensors konnte ermittelt werden, dass lediglich das *Streamen der Bilddaten* möglich ist.

Konfiguration 3:

- Aus der Plattformbeschreibung konnte geschlossen werden, dass die *Missionsdauer maximal 50 Minuten* beträgt.

- Aus der Wahl des Mountings konnte ermittelt werden, dass der Sensor *stabilisiert* wird und in *alle Richtungen bewegt* werden kann. Zudem steht der Zugriff auf einen *Geolock* zur Verfügung.
- Aus der Kombination der Computerhardware und dem Sensor wurde ermittelt, dass die *Detektion* von *Personen* und *Fahrzeugen* nur bei *Tag* durchgeführt werden kann. Als Algorithmus wurde der *DPM Classifier* verwendet.

Mit den Experimenten wurde gezeigt, wie die Ontologie aus zwei unterschiedlichen Betrachtungen (aus Missions- und aus UAV-Sicht) korrekte Aussagen bereitstellt. Aus einem Bestand von UAVs, deren Ressourcen und Missionsbeschreibungsparametern wurden geeignete UAV-Konfigurationen bzw. in Frage kommende Missionen ermittelt.

Im Falle der ermittelten UAV-Konfigurationen wurden zum Teil mehrere alternative Konfigurationen (vgl. Tabelle 21) gefunden, wodurch es notwendig wurde, anhand weiterer Parameter (z.B. Kosten) manuell zu entscheiden, welche der geeigneten Konfigurationen tatsächlich für die Mission eingesetzt werden soll. Ziel der Experimente war die Ermittlung geeigneter UAV-Konfigurationen anhand von Missionsbeschreibungen. Dieses wurde erfolgreich demonstriert.

Folgende Empfehlungen zur Erweiterung des Modells können aus den Resultaten abgeleitet werden:

- **Breite der Kategorieauswahl**:
  Neben den Missionsauswahlparametern wäre es z.B. denkbar diese um weitere Parameter zu ergänzen (z.B. Einsatzradius bzw. Entfernung und Einsatzhöhe). Der Fokus in dieser Arbeit lag auf den perzeptiven Fähigkeiten, jedoch wäre auch ein Fokus, der mehr auf den Charakteristika des UAVs oder der Sensoren liegt, denkbar.
- **Mehrere Auswahlmöglichkeiten**:
  Eine Mehrfachauswahl an Methoden (z.B. detektieren und aufzeichnen) war in dem Modell nicht vorgesehen. Dadurch konnte der Benutzer nicht zwei Methoden gleichzeitig verwenden, sondern musste zwei Aufgaben formulieren, damit diese zeitgleich durchgeführt werden. Es wäre denkbar, das Modell derart zu erweitern, dass solche Auswahlmöglichkeiten möglich sind.
- **Vage Formulierungen**:
  In der Ontologie kann die aktuelle Wetterlage, z.B. „*Rainy*“ und „*Sunny*“ in der Kategorie „*Weather Constraints*“ beschrieben werden. Die Beschreibung hat Einfluss auf die Wahl einer Perzeptionskette. Was jedoch genau „*Rainy*“ bedeutet und ab wann das Wetter als „*Rainy*“ eingestuft wird, kann mit dem Modell nicht beschrieben werden. Hier wäre ein genaueres Model notwendig, welches das Wetter in einem höheren Detailgrad beschreibt.
- **Modellierungstiefe**:
  Im bestehenden Modell konnte keine komplexe UAV-Modellierung durchgeführt werden (z.B. zwei Gimbals an einem UAV). Es konnte auch keine Aussage z.B. über die Montagepositionen des Gimbal / Sensor getroffen werden. Dies spielt im Falle von Sensoren eine wichtige Rolle, da hiervon die Ausrichtung einer Kamera abhängig sein kann.

- **Plug & Play**:
  Das hier entwickelte Modell der Ressourcen setzte voraus, dass die Ressourcen weitgehend modular via Plug & Play in verschiedenen Kombinationen genutzt werden konnten. In der Realität haben die meisten UAVs starre Konfigurationen mit fest definierten Sensoren und Sensorpositionen, wodurch das Tauschen von Hardware mit einem höheren Aufwand verbunden ist.

7.2.2 Missionsdurchführung

Bei den folgenden Experimenten wurde untersucht, ob und wie die Ontologie zur Zeit der Missionsdurchführung zur Erledigung von perzeptiven Aufgaben genutzt werden kann.

7.2.2.1 Untersuchungsgegenstand

Um zu validieren, wie sich das Softwaresystem und die Ontologie während einer Mission verhalten, wurde in einer Simulationsumgebung (siehe7.2.2.2) ein Szenario mit drei verschiedenen Aufklärungsaufgaben generiert. Die drei Aufgaben wurden mit dem UAV *Sagitta (vgl.* Kapitel 6.2.5) durchgeführt und dienten dem Zweck, gezielt verschiedene Fähigkeiten des UAV einzusetzen. Es wurde gezeigt, dass das System an Bord des UAV in der Lage ist, aufgabenspezifisch vorhandene Fähigkeiten auszuwählen und auf Veränderungen, wie z.B. Tageszeit oder den Ausfall von Ressourcen, zu reagieren. Um die Dynamik und das Reaktionsvermögen des Softwaresystems in Bezug auf Veränderungen zu testen, wurden dabei folgende Parameter an der Umgebung und dem UAV verändert:

- In der Simulationsumgebung wurde zwischen verschiedenen Uhrzeiten gewechselt, sodass eine Aufklärungsaufgabe sowohl bei Tag als auch bei Nacht durchgeführt werden konnte.
- Im UAV wurde ein Hardwareausfall simuliert, indem nacheinander die unterschiedlichen Computer an Bord des UAV deaktiviert wurden.

7.2.2.2 Methode und Experimente

Mit den Aufgaben der simulierten Mission wurde gezeigt, wie das Softwaresystem anhand der durchzuführenden Aufgabe und der sich verändernden Umweltparameter eine geeignete Perzeptionskette auswählte. Bevor darauf eingegangen wird, wie dies erfolgt ist, wird die Simulationsumgebung vorgestellt.

Simulationsaufbau

Als Referenzplattform diente das Experimental-UAV SAGITTA von Airbus Defence Space (vgl. Abbildung: 109), ein strahlengetriebenes, unbemanntes Flächenluftfahrzeug.

Abbildung: 109 Experimental-UAV SAGITTA

Folgende Missionssensoren sind für diese Plattform vorgesehen:

- Elektrooptisch: 2 x Prosilica GT mit fester Brennweite und einer Full HD Auflösung (1920 x 1080 Pixel).
- Elektrooptisch: Sony FCB mit einem 100mm zoombaren Objektiv, 30 FPS und einer Full HD Auflösung (1920 x 1080 Pixel).
- Infrarot: FLIR Tau 640 mit einer festen Brennweite, 30 FPS und einer Auflösung von 640x480 Pixel.

Während die Prosilica Sensoren fest in Flugrichtung verbaut sind, sind die beiden eigentlichen Missionssensoren (Sony FCB und FLIR Tau) in einem 3-Achsen Gimbal auf der Unterseite des UAVs integriert. Dadurch können die Sensoren unabhängig von der Fluglage für Aufklärungszwecke genutzt werden. Als Missionscomputer kommt ein sonderangefertigtes Compact PCI System (vgl. Abbildung 110) zum Einsatz.

Abbildung 110: Missionscomputer

Der Missionscomputer ist mit folgenden Komponenten ausgestattet:

- 2 x Intel i7 (CPU) Prozessorboards: (zur Missions- und Flugführung sowie zur Durchführung von Perzeptionsaufgaben mittels des SPMS),

- 1 x FPGA XLINX Prozessorboards,
- 1 x Motorola Prozessorboards (zur Kommunikation mit anderen Bordsystemen),
- 1 x Netzwerk Switch,
- 1 x Power Supply.

Die Bodenkontrollstation mit dem Sensor- und Kartendisplay ist in Abbildung 111 dargestellt.

Abbildung 111: Bodenkontrollstation

Als Simulationswerkzeug kam *Virtual Battle Space* (VBS) [62] zum Einsatz. VBS wird sowohl zur Simulation der Flugdynamik der UAV-Plattform, der Aufklärungssensoren (EO und IR), der Sensorhalterung (Gimbal), als auch zur Simulation der Umwelt (Wetter, Tageszeit, Fahrzeug- und Personenbewegung etc.) verwendet. Die Realisierung in der Simulationsumgebung resultierte in folgender Architektur (Abbildung 112):

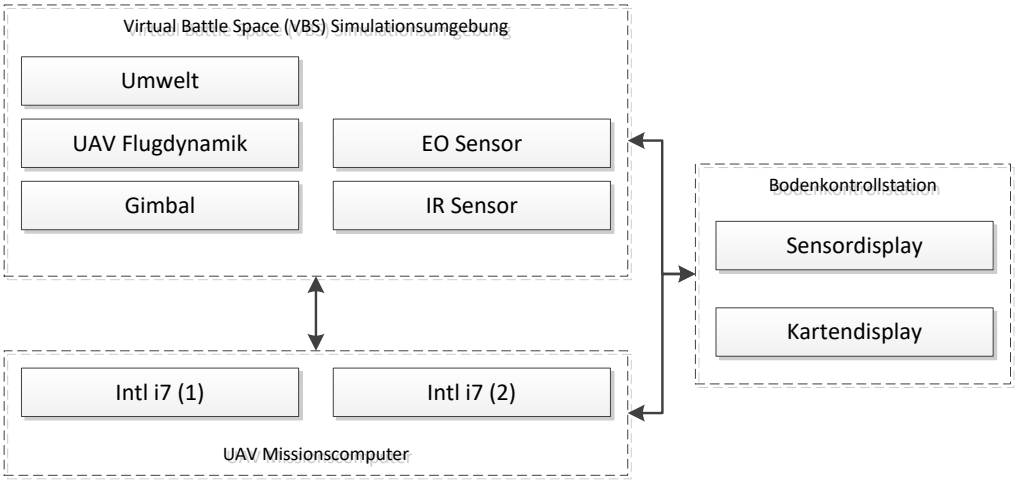


Abbildung 112: Hardware-Simulationsaufbau

Da die starr ausgerichteten Prosilica Sensoren des Experimental-UAVs für die Experimente keine Relevanz hatten, wurden diese in der Simulationsumgebung nicht nachgebildet. Für das

Experiment wurde die in der Abbildung 111 dargestellt Bodenkontrollstation, bestehend aus zwei Rechnersystemen und Bildschirmen, verwendet. Der Missionscomputer stand für die Experimente nicht zur Verfügung, weshalb dieser auf zwei handelsüblichen Intel i7 CPUs (*Intel i7 (1)* und *Intel i7 (2)*) simuliert wurde. Zusätzlich wurde für einzelne Algorithmen angenommen, dass diese nur auf spezifischer Hardware (FPGA und GPU) lauffähig sind.

Missionsablauf

Der Gesamtablauf der simulierten Mission ist in Abbildung 113 dargestellt. Im südlichen Teil befand sich der befreundete Flughafen, von dem aus das UAV die Mission begann. Nach dem Passieren des Korridors (blau, „*Corridor B*“) observierte das UAV zunächst eine befreundete, vorgelagerte Radarstation (A). Im Anschluss wurde eine Truppenbewegung im nördlichen Teil des Gebietes aufgeklärt (B). Zuletzt klärte das UAV eine militärische Basis (C) auf, bevor es zum Ausgangspunkt zurückkehrte.

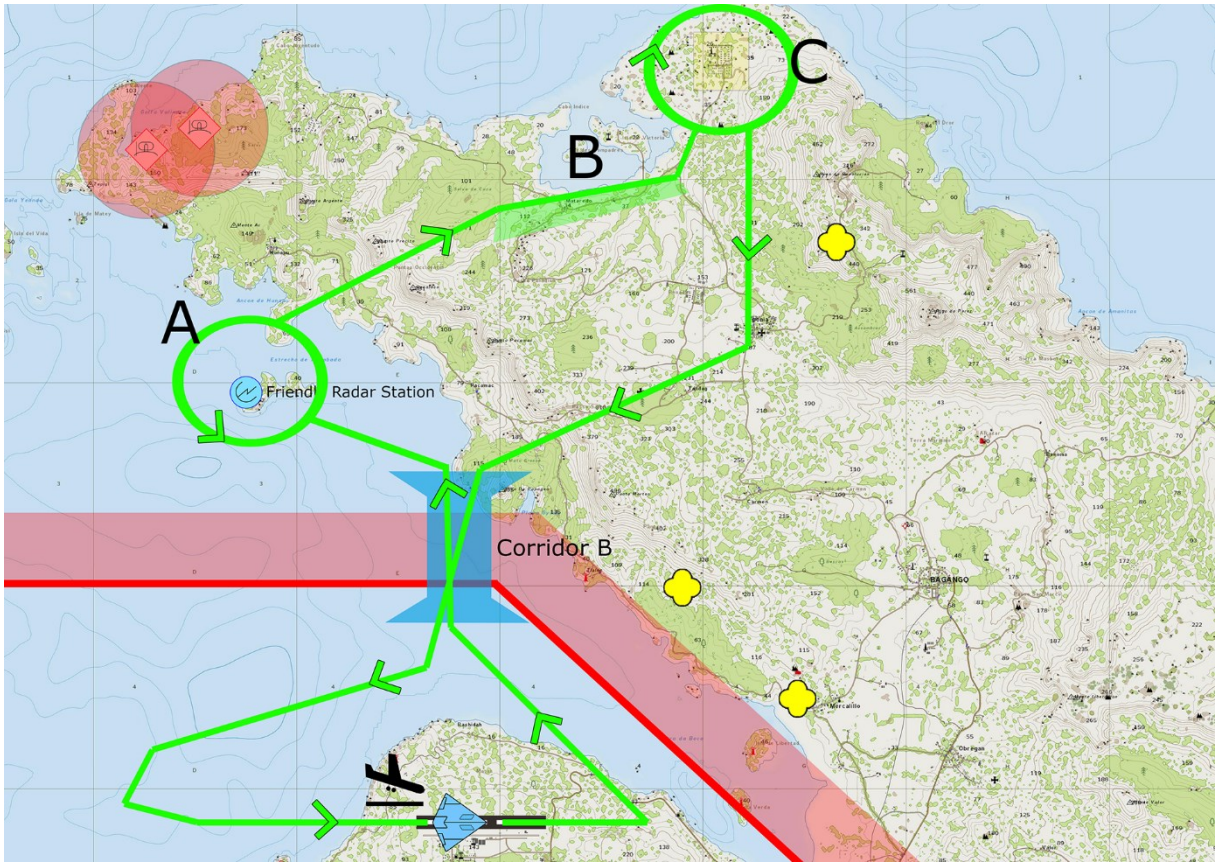


Abbildung 113: Im unteren Teil befindet sich der Startflughafen. In Grün ist die zu fliegende Route des UAVs zu sehen. Die verschiedenen Aufgaben der Mission sind mit den Buchstaben A, B und C markiert.

Die Mission war in drei Aufgaben A, B und C gegliedert. Die Aufgaben waren nacheinander vom UAV durchzuführen.

Aufgabe A

Bei dieser Aufgabe sollten die Sensordaten an die Bodenkontrollstation gestreamt werden. Dem Benutzer stand es frei, zu entscheiden, ob er sich den Videostream des EO-Sensors anzeigen lässt oder den des Infrarotsensors.

- Alle UAV-Systeme waren voll funktionsfähig.
- Die Umweltbedingungen waren: Wetter: Klar, Missionszeit:12:00-13:00 Uhr.

Die Beschreibung für diese Aufgabe, um die relevanten Fähigkeiten und Ressourcen aus der Ontologie zu extrahieren, sah wie folgt aus:

$$Aufgabe_A = \left\{ \begin{matrix} \boldsymbol{Methode}: \boldsymbol{Streaming}, \\ \boldsymbol{Objekt}: -, \\ Typ: -, \\ Ort: -, \\ Tageszeit: - \end{matrix} \right\}$$

Lediglich die Methode „Streaming“ wurde festgelegt, alle anderen Parameter wurden nicht vorgegeben.

Aufgabe B

Bei dieser Aufgabe wurden erste Automatisierungsfunktionen des UAVs untersucht. Es sollten mehrere Fahrzeuge auf einer Straße detektiert, annotiert, mit Metainformationen (GPS-Position, Zeit der Erkennung, Kategorisierung etc.) versehen und die Daten als Ergebnisse (versehen mit einem militärischen SIDC-Code) an die Bodenkontrollstation übertragen werden. Der Operateur konnte sich die einzelnen Ergebnisse im Detail betrachten und diese analysieren (z.B. Ermitteln der Fahrtrichtung aufgrund der Fahrzeugorientierung).

- Die GPU und das FPGA des UAV-Systems wurden als nicht funktionsfähig simuliert.
- Die Umweltbedingungen waren: Wetter: Klar, Missionszeit: 01:00-02:00 Uhr.

Die Aufgabe, um die relevanten Fähigkeiten und Ressourcen aus der Ontologie zu extrahieren, wurde wie folgt beschrieben:

$$Aufgabe_B = \left\{ \begin{matrix} \boldsymbol{Methode}: \boldsymbol{Detektion}, \\ \boldsymbol{Objekt}: \boldsymbol{Fahrzeug}, \\ Typ: -, \\ Ort: -, \\ Tageszeit: - \end{matrix} \right\}$$

Die Methode „Detektion“ und das Objekt „Fahrzeug“ wurden definiert.

Aufgabe C

Hier sollten mittels eines alternativen Bildverarbeitungsalgorithmus Fahrzeuge innerhalb einer militärischen Basis detektiert und annotiert werden.

- Das FPGA des UAV-Systems wurde als nicht funktionsfähig simuliert.
- Die Umweltbedingungen waren: Wetter: Bewölkt, Missionszeit: 14:00-15:00 Uhr

Die Aufgabe war wie folgt beschrieben:

$$Aufgabe_C = \left\{ \begin{matrix} \boldsymbol{Methode}: \boldsymbol{Detektion}, \\ \boldsymbol{Objekt}: \boldsymbol{Fahrzeug}, \\ Typ: -, \\ Ort: -, \\ Tageszeit: - \end{matrix} \right\}$$

- Die Beschreibung in Aufgabe C ist identisch mit der Beschreibung in Aufgabe B.

Die freien Parameter für die drei Aufgaben sind in Tabelle 25 zusammengefasst.

		Aufgabe A	Aufgabe B	Aufgabe C
Freie Para-meter	**Methode**	Streaming/Recording	Detektion	Detektion
	Entität	-	Fahrzeuge	Fahrzeuge
	Hardwareausfall	-	GPU, FPGA	FPGA
	Uhrzeit	12:00-13:00	01:00-02:00	14:00-15:00
	Wetterlage	Klar	Klar	Bewölkt

Tabelle 25: Freie Parameter der drei Aufgaben

Sie sind gegliedert in

- Aufgabenspezifisch: Methode (z.B. *Streaming*) und Entität (z.B. *Fahrzeug*),
- UAV-System: Hardwareausfall (z.B. *FPGA*),
- Umwelt: Uhrzeit (z.B. *01:00-02:00*) und Wetterlage (z.B. *Bewölkt*).

Die aufgabenspezifischen Parameter waren aufgrund der Aufgabenstellung vorgegeben. Die Parameter des UAV-Systems und die der Umwelt ergaben sich aus der definierten Aufgabenstellung.

7.2.2.3 Ergebnisse

In Tabelle 26 sind die resultierenden abhängigen Variablen, die sich aus den freien Parametern ergeben, für die drei Aufgaben zusammengefasst.

Die freien Parameter wurden bereits in Tabelle 25 erläutert. Aus ihnen ermittelte das Softwaresystem abhängige Variablen bzw. Kennzahlen, die den Systemzustand widerspiegelten. Zu diesen zählten:

- Sensorwahl: ergab sich u.a. aus der Uhrzeit
- Anzahl der vorhandenen Bildverarbeitungsmodule (BV-Module): ergab sich aus der funktionsfähigen Hardware
- Anzahl der nutzbaren BV-Module: ergab sich aus der Kombination aus Sensor, Hardware und den BV-Modulen
- Anzahl der Fähigkeiten: ergab sich aus der Kombination der BV-Module
- Anzahl der BV-Module in einer Kette: ergab sich aus der gewählten Fähigkeit
- Detektierungsmethode: ergab sich aus der Kombination des Sensors, der Aufgabe und der Fähigkeiten

		Aufgabe A	Aufgabe B	Aufgabe C
Freie Parameter	**Methode**	Streaming/Recording	Detektion	Detektion
	Entität	-	Fahrzeuge	Fahrzeuge
	Hardwareausfall	-	GPU, FPGA	FPGA
	Uhrzeit	12:00-13:00	01:00-02:00	14:00-15:00
	Wetterlage	Klar	Klar	Bewölkt
Abhängige Variablen	**Sensorwahl**	EO	IR	EO
	# vorhandener BV-Module	22	15	20
	# nutzbarer BV-Module	19	11	17
	# Fähigkeiten	7	6	6
	# BV-Module in Kette	2	4	6
	Detektierungsmethode	RGB Streaming	IR Blob Detection	DPM Classifier

Tabelle 26: Freie Parameter und resultierenden Kennzahlen der Ressourcen

Aufgabe A:

Die erste Aufgabe diente dem Zweck, den initialen (vollfunktionsfähigen) Zustand des UAVs zu Missionsbeginn zu verifizieren. Die Umweltbedingungen waren klares Wetter bei Tag. Alle installierten Ressourcen waren vollfunktionsfähig. Aufgrund dessen hatte das System Zugriff auf 22 BV-Module. Zwei Module davon waren zwar installiert, konnten jedoch aufgrund fehlender Daten nicht benutzt werden, da es sich um Algorithmen, die auf Basis von SAR-Sensor-Daten arbeiten, handelte. Aus den Perzeptionsketten, die mit den verbleibenden 19 Modulen erstellt werden konnten, ergaben sich sieben Fähigkeiten. Von diesen war nur *Streaming* im Kontext der Aufgabe angefragt. Der Perzeptionsgraph, mit dem alle Ketten zum Streamen von Daten identifiziert werden konnten, ist in Abbildung 114 zusehen.

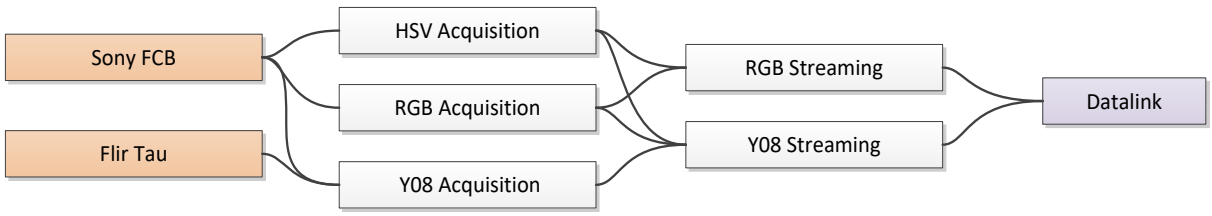


Abbildung 114: Perzeptionsgraph für die Fähigkeit *Streaming*

Aus dem Graph konnten die in Tabelle 27 dargestellten möglichen Ketten ermittelt werden.

Nr.	Sensor	Modul	Modul	Datenlink
1	Sony FCB	HSV Acquisition	RGB Streaming	Datalink
2	Sony FCB	HSV Acquisition	Y08 Streaming	Datalink
3	**Sony FCB**	**RGB Acquisition**	**RGB Streaming**	**Datalink**
4	Sony FCB	RGB Acquisition	Y08 Streaming	Datalink
5	Sony FCB	Y08 Acquisition	Y08 Streaming	Datalink
6	Flir Tau	Y08 Acquisition	Y08 Streaming	Datalink

Tabelle 27: Perzeptionsketten für die Aufgabe A

Von diesen hat das System die Fähigkeit *RGB Streaming* (Nr. 3) ausgewählt. Die Perzeptionskette bestand aus zwei Modulen (vgl. Abbildung 115). Aufgrund der Perzeptionskettenauswahl wurde in diesem Fall der EO-Sensor verwendet. Die Daten wurden über einen Datenlink an die Bodenkontrollstation übertragen.

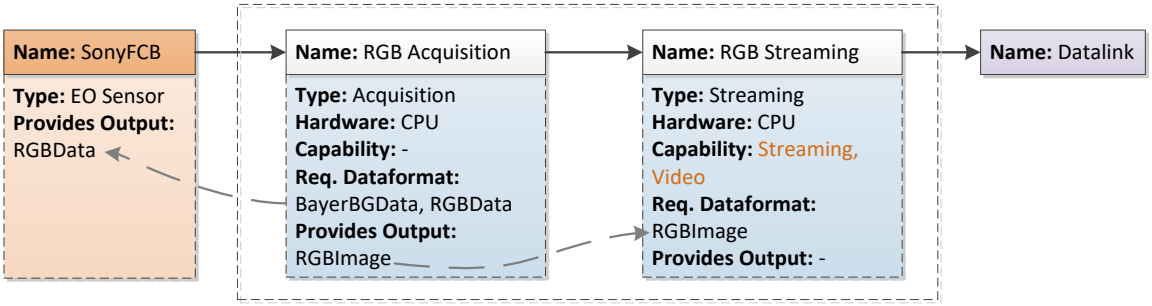


Abbildung 115: Perzeptionskette zur Übertragung von RGB Bilddaten

Ein Ausschnitt des übertragenen Bilddatenstroms ist in Abbildung 116 zu sehen.

Abbildung 116: Ergebnis aus Aufgabe A: Streaming

Aufgabe B:

Ziel dieser Aufgabe war es, Fahrzeuge automatisiert zu detektieren. Die Umweltbedingungen waren klares Wetter bei Nacht. Es wurde der Hardwareausfall von GPU und FPGA simuliert.

Daraus folgend konnten nur die 15 BV-Module betrieben werden, die auf einer CPU ausgeführt wurden. Von diesen 15 konnten 11 für eine geeignete Perzeptionskette genutzt werden. Für die übrigen vier konnten die Vor- bzw. Nachbedingungen nicht erfüllt werden. Zum Beispiel wäre es möglich gewesen, ROIs auf Basis von IR bzw. RGB-Daten bereitzustellen, jedoch gab es aufgrund des Hardwareausfalls keine Module, die diese Daten verarbeiten konnten. Aufgrund der Uhrzeit wurden nur Ketten beachtet, die mit Daten eines Infrarotsensors arbeiten können. Abbildung 117 zeigt den Graphen zu drei verschiedenen Zeitpunkten, um die Funktionsweise des Systems und die daraus resultierenden Systemveränderungen zu erläutern.

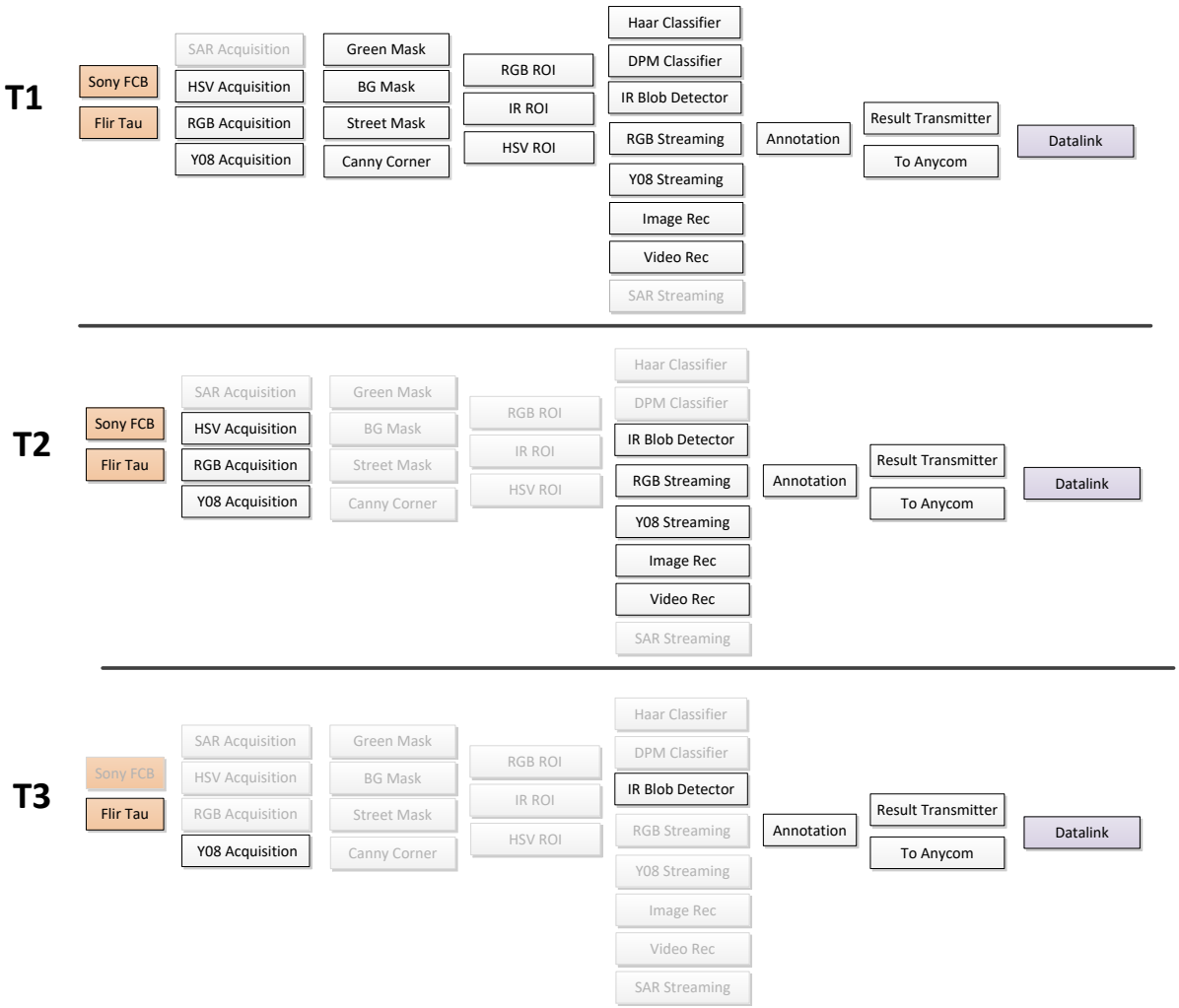


Abbildung 117: Der Graph zum Zeitpunkt dreier verschiedener Zustände

- Zum Zeitpunkt T1 befand sich das System in dem Zustand aus Aufgabe A. Alle Systeme waren funktionsfähig. Zwei BV-Module konnten nicht genutzt werden, weil die notwendigen Daten eines SAR Sensors fehlen.
- Zum Zeitpunkt T2 stellte das System fest, dass die GPU und der FPGA nicht mehr funktionsfähig sind. Deshalb wurden im Graphen alle Module, die diese Art der Hardware zur Ausführung benötigen, deaktiviert.

- Zum Zeitpunkt T3 erreichte das UAV die Position, an der die Aufklärungsaufgabe durchgeführt werden sollte. Aus der Aufgabenbeschreibung und den aktuellen Umweltbedingungen konnte das System schließen, dass die Daten vom EO-Sensor unbrauchbar sind und dass Fähigkeiten, die kein Fahrzeug erkennen, nicht benötigt werden.

Zum Zeitpunkt der Durchführung von Aufgabe B existierte der in Abbildung 118 dargestellte Graph.

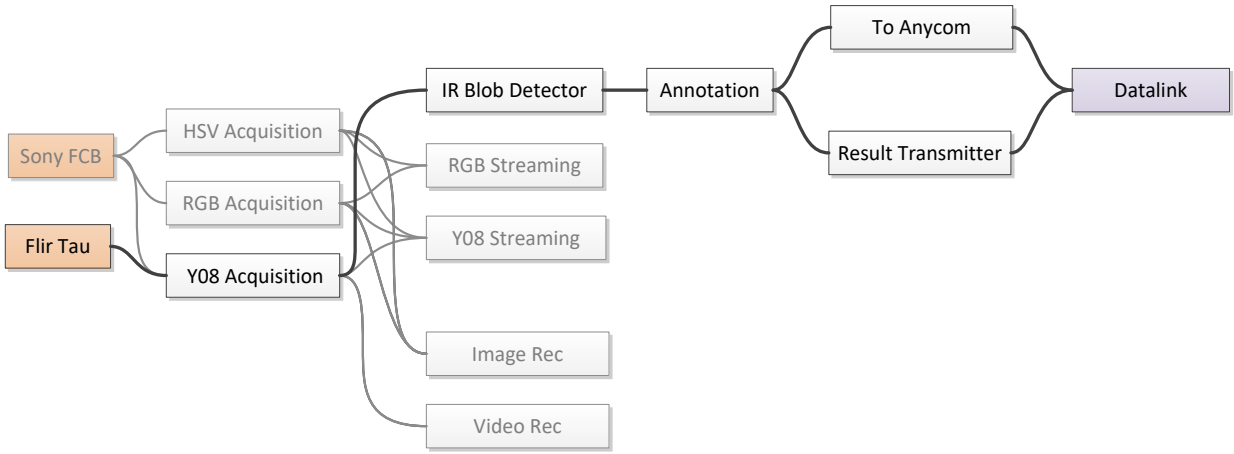


Abbildung 118: Perzeptionsgraph für die Aufgabe B

Die Ketten, die zwar aufgrund der vorhandenen Hardware ausgeführt werden konnten, jedoch wegen der Aufgabenbeschreibung oder der Umweltbedingungen nicht in Frage kamen, wurden deaktiviert (in der Abbildung grau dargestellt). Es blieben die in Tabelle 28 angegebenen Perzeptionsketten übrig.

Nr.	Sensor	Modul	Modul	Modul	Modul	Datenlink
1	Flir Tau	Y08 Acq.	IR Blob Detector	Annotation	To Anycom	Datalink
2	**Flir Tau**	**Y08 Acq.**	**IR Blob Detector**	**Annotation**	**Result Trans.**	**Datalink**

Tabelle 28: Perzeptionsketten für die Aufgabe B

Die vom System ausgewählte Kette (Nr. 2) bestand aus vier Modulen und nutzt zur Detektion das Modul *IR Blob Detection*. Die Kette ist in Abbildung 119 zu sehen.

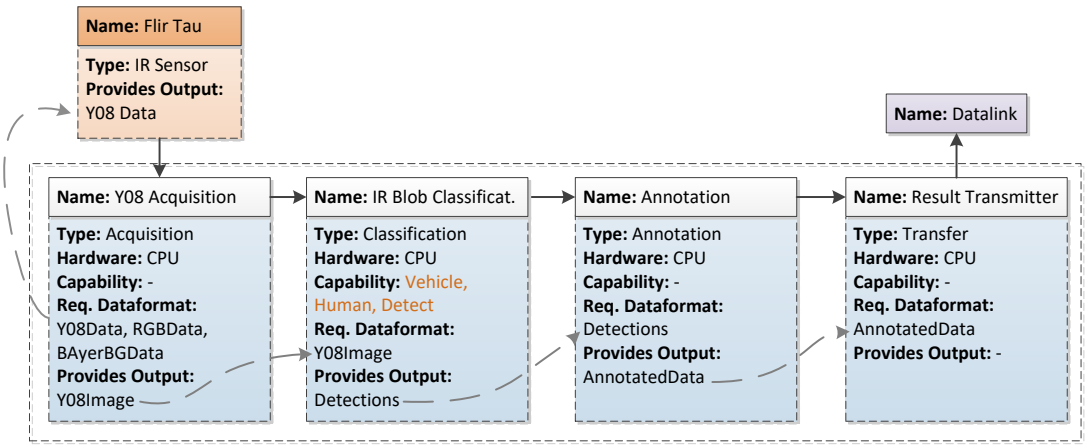


Abbildung 119: Perzeptionskette zur Erkennung von Fahrzeugen auf Basis von IR Daten

Die Abbildung zeigt, dass jedes BV-Modul auf einer CPU ausgeführt wird. Das Ergebnis der Detektion und der annotierten Daten im Kartendisplay ist in Abbildung 120 dargestellt.

Abbildung 120: Ergebnis der Fahrzeugdetektion für Aufgabe B

Aufgabe C:

In der Aufgabe C sollten Fahrzeuge in einer militärischen Basis detektiert werden. Es herrschte bewölktes Wetter bei Tag. Ein Hardwareausfall des FPGA wurde simuliert. Daraus folgend standen 20 BV-Module zur Verfügung. Für 17 davon waren die Vor- bzw. Nachbedingungen erfüllt. Nach Abschluss der Inferenz konnten folgende Perzeptionsketten aus dem in Abbildung 121 dargestellten Graphen ermittelt werden:

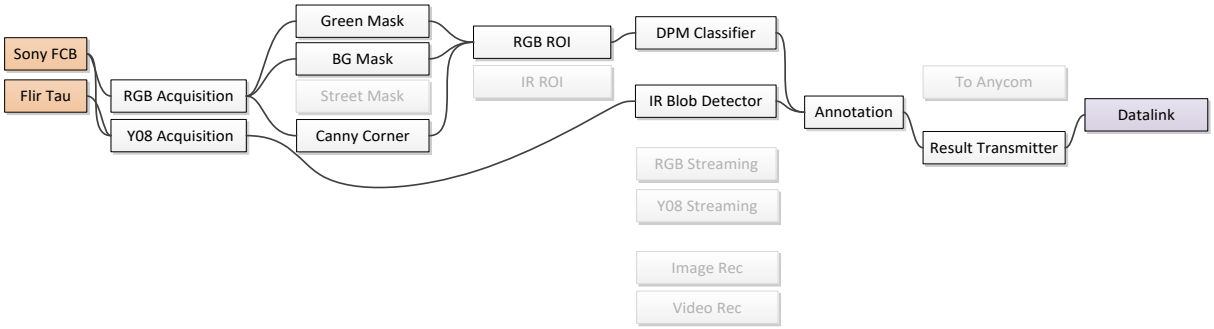


Abbildung 121: Perzeptionsgraph für Aufgabe C

Ketten, die wegen der Aufgabenbeschreibung oder der Umweltbedingungen nicht in Frage kamen, wurden deaktiviert (in der Abbildung grau dargestellt). Es resultierten die in Tabelle 29 angegebenen Perzeptionsketten.

Nr.	1	2	3	4
Sensor	**Sony FCB**	Sony FCB	Sony FCB	Flir Tau
Modul	**RGB Acq.**	Y08 Acq.	IR Blob Detector	Y08 Acq.
Modul	**Green Mask**	BG Mask	Canny Corner	IR Blob Detec.
Modul	**RGB ROI**	RGB ROI	RGB ROI	
Modul	**DPM Class.**	DPM Class.	DPM Class.	
Modul	**Annotation**	Annotation	Annotation	Annotation
Modul	**Result Trans.**	Result Trans.	Result Trans.	Result Trans.
Datenlink	**Datalink**	Datalink	Datalink	Datalink

Tabelle 29: Perzeptionsketten für die Aufgabe C

Die vom System ausgewählte Kette (Nr. 1) bestand aus sechs Modulen und benutzte zur Detektion das Modul *DPM Classification*. Dieses Modul nutzt ein zuvor trainiertes Modell von Fahrzeugen, wodurch die Erkennungsrate im Vergleich zur Erkennung z.B. auf Basis von IR Daten deutlich höher ist. Die Kette ist in Abbildung 122 zusehen.

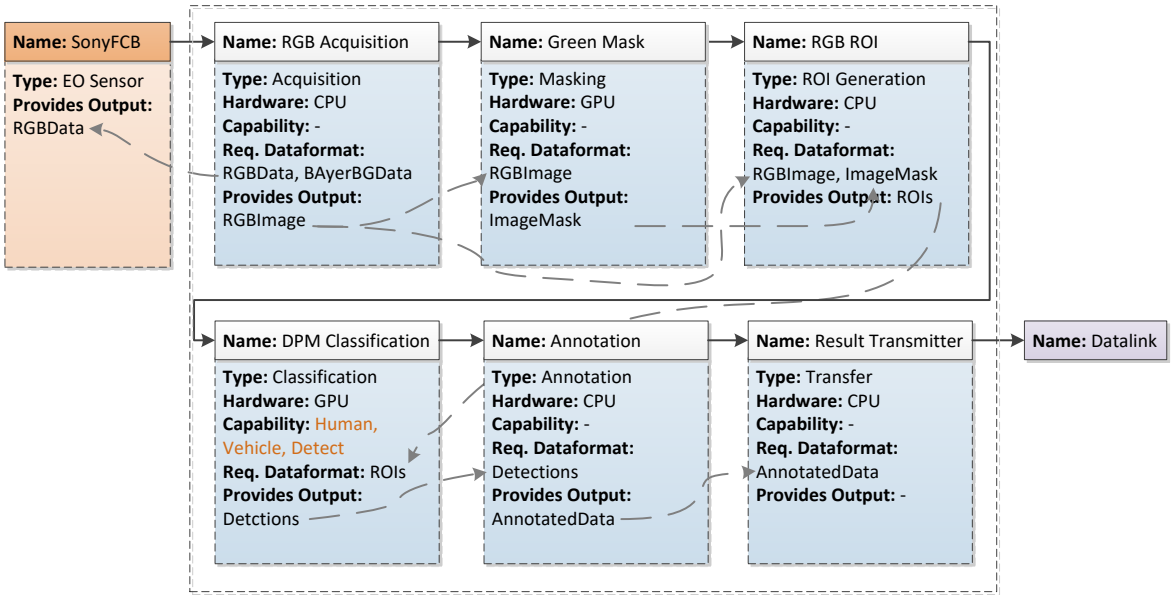


Abbildung 122 Perzeptionskette zur Erkennung von Fahrzeugen mittels eines trainierten Klassifikators

In der Abbildung ist zu sehen, dass die Module *DPM Classification* und *Green Mask* auf einer GPU ausgeführt wurden, während der Rest auf einer CPU lief. Das Ergebnis der Detektion und der annotierten Daten im Kartendisplay ist in Abbildung 123 dargestellt.

Abbildung 123: Ergebnis der Fahrzeugerkennung für Aufgabe C

Die Experimente in diesem Anwendungsfall zeigten, wie das Softwaresystem auf Basis der Umweltbedingungen, des Zustands des UAV-Systems und der Aufgabenstellung passende Perzeptionsketten auswählte, sodass die Aufgabe erfolgreich durchgeführt werden konnte. Hardwareausfälle wurden während der Missionsdurchführung detektiert und die Auswirkungen der Ausfälle dem System mitgeteilt, sodass dieses entsprechende Maßnahmen bzw. Rekonfigurationen von Perzeptionsketten durchführen konnte.

Ziel der Experimente war es, zu untersuchen, wie verschiedene Perzeptionsketten zustande kommen. Die Experimente zeigten, dass für das erfolgreiche Durchführen einer Aufgabe mehrere Perzeptionsketten zur Auswahl standen. Die Methode, wie die Auswahl stattgefunden hat, ist Bestandteil einer anderen Forschungsarbeit. Erste Ergebnisse zur Auswahlmethode können in [61] eingesehen werden.

7.2.3 Fazit der operationellen Bewertung

Bei den Experimenten der operationellen Bewertung wurde untersucht, wie die Ontologie und das entwickelte Softwaresystem dazu beitragen, die Aufgabenstellung aus Kapitel 1.4 zu erfüllen. Die Experimente wurden zunächst für den Einsatz zur Missionsvorbereitung durchgeführt. Die anschließenden Experimente untersuchten den Einsatz während der Missionsdurchführung.

Bei dem Einsatz zur Missionsvorbereitung wurde gezeigt, dass die Ontologie auf zwei Arten genutzt werden kann. In einem *Top-Down*-Ansatz wurde in drei verschiedenen Missionen erfolgreich demonstriert, dass, ausgehend von der Missionsbeschreibung, die Ontologie dafür eingesetzt werden kann, zu ermitteln, welche UAV-Konfigurationen für die spezifische Missionsaufgabe in Frage kommen. Die Ergebnisse des *Bottom-Up* Ansatzes zeigten, dass, ausgehend von drei unterschiedlichen UAV-Konfigurationen, mittels der Ontologie korrekt ermittelt werden konnte, welche perzeptiven Aufgaben mit den jeweiligen UAV-Konfigurationen durchgeführt werden können.

Während der Missionsdurchführung wurde demonstriert, wie ein Softwaresystem die Ontologie nutzten konnte, um automatisiert verschiedene perzeptive Aufgaben durchzuführen. Hierzu wurden im Rahmen einer Mission drei unterschiedliche Aufgaben gestellt. Um die Reaktionsfähigkeit auf unvorhergesehene Ereignisse wie Hardwareausfall und sich ändernde Wetterbedingungen zu demonstrieren, wurden entsprechende szenarische Elemente simuliert. Anhand von drei Aufgaben wurde in den Experimenten erfolgreich nachgewiesen, dass das Modell an Bord eines UAVs zur automatisierten Durchführung von Aufklärungsmissionen in einem dynamischen Missionskontext eingesetzt werden kann. Es wurde gezeigt, dass das Softwaresystem designgemäß auf Ereignisse im Missionsablauf, wie z.B. einen Hardwareausfall, reagiert und unter Verwendung der Ontologie eine alternative Lösungsstrategie erfolgreich entwickelt.

7.3 Leistungsbewertung

In Kapitel 3.4 wurde die Einhaltung einer begrenzten Verarbeitungszeit gefordert, um das Softwaresystem in einer dynamischen Mission zu nutzen. Im Rahmen der Experimente wurde keine präzise Anforderung an eine Abfrage in Realzeit gestellt, jedoch sollten das Ermitteln von Veränderungen im System oder in der Umwelt und die daraus folgenden Schlussfolgerungen im Sekundenbereich liegen. In der Leistungsbewertung wurden die Antwortzeiten der Ontologie für unterschiedliche Anfragen untersucht.

7.3.1 Untersuchungsgegenstand

Für die Untersuchung der Abfragezeiten wurde gemessen, wie lange es dauerte, spezifische Informationen aus der Ontologie auszulesen. Hierzu wurden zwei Ansätze untersucht:

- *OWL Referenzimplementierung* (vgl. Kapitel 6.5): Hierbei wird auf eine Datenbank zum Speichern und Laden der Ontologie verzichtet. Stattdessen werden die Informationen direkt in einer Datei auf die Festplatte gespeichert bzw. von dieser geladen.
- *Tripelstore (RDF Store)* (vgl. Kapitel 6.3)*:* Die Informationen werden in einer Datenbank in Form von Tripel (Subjekt-Prädikat-Objekt) abgelegt und mittels SPARQL (vgl. Kapitel 5.3.2.4) abgefragt.

Die Abfragezeiten der beiden Ansätze wurden verglichen, um zu ermitteln, mit welchem Ansatz kürzere Abfragezeiten erreicht werden konnten. Ziel der Untersuchung war es, eine Aussage darüber treffen zu können, ob und wann die Referenzimplementierung bzw. der Tripelstore eingesetzt werden sollte.

7.3.2 Methode und Experimente

Für dieses Experiment wurde die Ontologie aus Kapitel 6.2 benutzt. Die Messungen fanden auf einem AMD Phenom II X4 955 (3,20 GHz, 8 GB RAM) statt. Das System ist ein x64 Windows 10 Pro N. Als Festplatte kam eine Samsung SSD zum Einsatz. Die Zeitmessung wurde mit Java-internen Bibliotheken durchgeführt.

Da der Tripelstore direkt via SPARQL abgefragt wurde, für die Referenzimplementierung jedoch zuerst passenden Methoden implementiert werden mussten, wiesen die Abfragen strukturelle Unterschiede auf. Der Informationsgehalt des Ergebnisses der Abfragen war jedoch identisch.

Im Folgenden werden die unterschiedlichen Experimente erläutert:

7.3.2.1 Inferenzdauer

Es wurde die Zeitdauer gemessen, bis die Inferenzmaschine (in beiden Fällen Hermit) auf implizites Wissen schließen konnte. Da in einer dynamischen Mission der Zustand der Ontologie kontinuierlich variiert und sich dadurch Vor- und Nachbedingungen sowie andere Bedingungen ständig ändern, muss die Inferenzmaschine regelmäßig implizites Wissen neu aufbauen. Im nächsten Experiment wurde die Dauer einer Abfrage für beide Ansätze untersucht.

7.3.2.2 Abfragedauer

Die Abfragezeiten sind maßgeblich von der Größe und Komplexität der Ontologie abhängig. Damit ein möglichst großer Teil der Ontologie traversiert werden konnte und um möglichst

viele Informationen mit einer einzigen Abfrage zu ermitteln, wurden für die Instanzen verschiedener Klassen alle Informationen, die mittels *Objectproperties*, *Dataproperties* und Klassenzugehörigkeit gefunden wurden, ausgegeben. Für eine Plattform wären dies also alle gültigen Konfigurationen und die dazugehörigen Fähigkeiten. In Abbildung 124 ist der generelle Ablauf einer Abfrage illustriert.

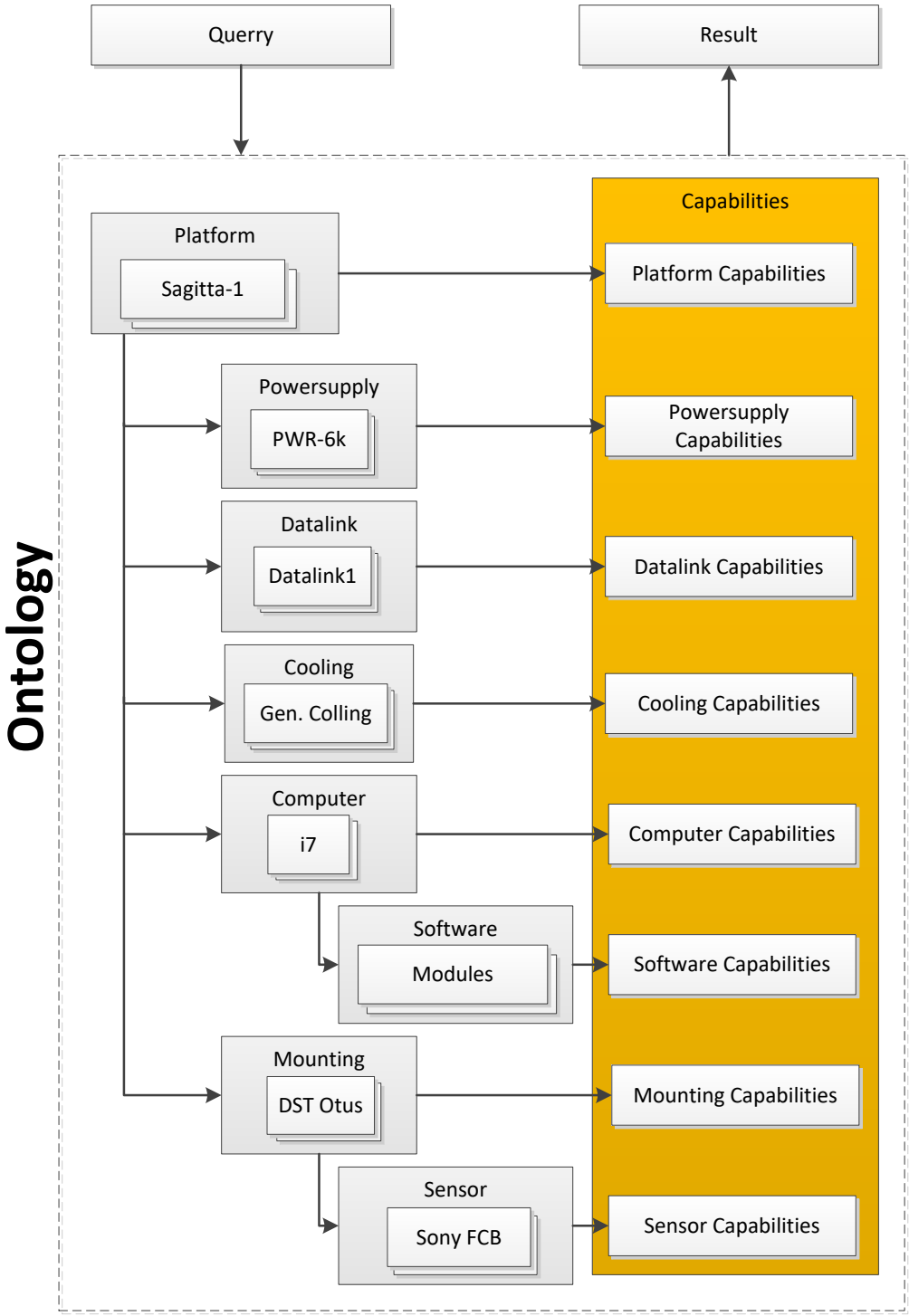


Abbildung 124: Ermittelte Informationen einer Abfrage aus der Ontology

Über die Abfrage wird nach den Konfigurationen für eine Plattform gefragt. Anhand der Plattformspezifikation wird dann ermittelt, welche Ressourcen mit der Plattform kombiniert werden können. Zusätzlich werden die zugehörigen Fähigkeiten ermittelt. Existieren für ein UAV z.B. keine kombinierbaren Sensoren und keine Computer, werden die dazugehörigen Klassen in der Ontologie übersprungen, wodurch sich die Abfragezeit verkürzt.

Die Abfragen für dieses Experiment sind im Anhang unter (A) zu finden. Abhängig von der Komplexität der Klasse mussten verschiedene Abfragen für eine Klasse durchgeführt werden.

7.3.2.3 Gesamtdauer

Bei diesem Experiment wurde schließlich untersucht, wie viel Zeit von einer Systemveränderung bis zum Ergebnis einer Abfrage verstrich. Zunächst musste dazu der Inferenzmechanismus gestartet und anschließend das Ergebnis einer Abfrage zurückgegeben werden.

Die Abfragen für dieses Experiment sind im Anhang unter (B) zu finden.

7.3.3 Ergebnisse

7.3.3.1 Inferenzdauer

In Diagramm 2 ist die Dauer der Inferenz für den Tripelstore und die Referenzimplementierung in Millisekunden abgebildet. Der Inferenzprozess bei der Referenzimplementierung bestand aus zwei Teilen: Vorverarbeitung und Anlegen der Ontologie.

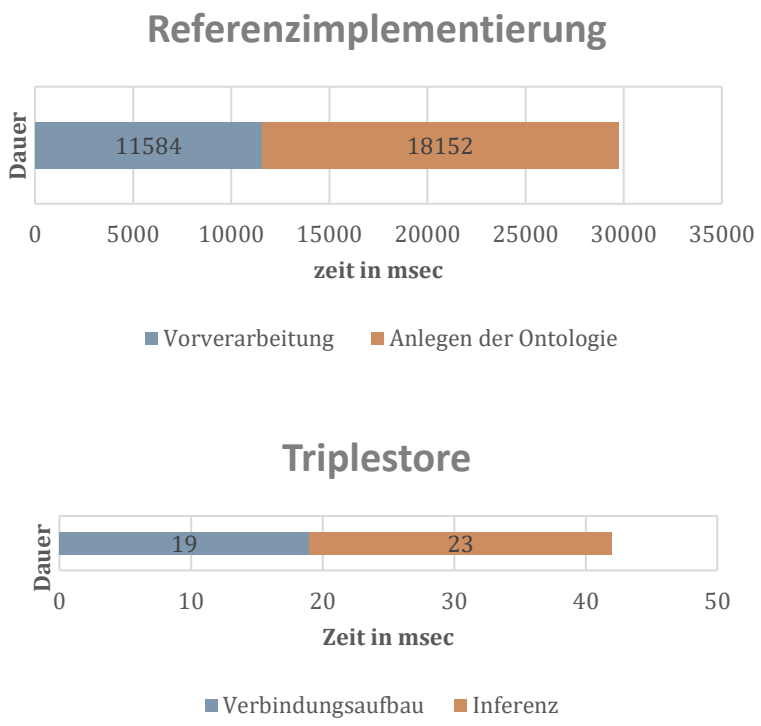


Diagramm 2: Inferenzdauer bei der OWL Referenzimplementierung und dem Tripelstore

Der Inferenzprozess besteht aus dem Verbindungsaufbau und der eigentlichen Inferenz. Die Dauer für die Inferenz beim Tripelstore beträgt lediglich **42 Millisekunden**. Dies liegt daran, dass

- der Inferenzmechanismus in der Datenbank ständig aktiv ist und automatisch bei einer Veränderung die Auswirkungen innerhalb der Beziehung der Daten ermittelt und
- der Inferenzmechanismus innerhalb des Tripelstores optimiert wurde.

Bei der Referenzimplementierung wird die OWL Datei von einem Datenspeicher geladen. Um implizites Wissen zu erhalten, muss der Inferenzmechanismus gestartet werden, sobald eine Veränderung festgestellt wurde. Der Prozess fand in zwei Schritten statt:

1. Vorverarbeitung: Dabei wird festgelegt, welche Axiome (vgl. Listing 14) inferiert werden sollen. Die Anzahl der Eigenschaften bestimmt dabei die Dauer des Vorgangs.
2. Anlegen der Ontologie: Es wird eine neue Ontologie im Speicher angelegt. Diese beinhaltet sowohl das explizite Wissen aus der geladenen Datei als auch das implizite Wissen, das über die Inferenzmaschine bestimmt wurde.

```
1  hermit = factory.createReasoner(ontology);
2  hermit.precomputeInferences(InferenceType.CLASS_HIERARCHY);
3  hermit.precomputeInferences(InferenceType.OBJECT_PROPERTY_HIERARCHY);
4  hermit.precomputeInferences(InferenceType.DATA_PROPERTY_HIERARCHY);
5  hermit.precomputeInferences(InferenceType.OBJECT_PROPERTY_ASSERTIONS);
6  hermit.precomputeInferences(InferenceType.DATA_PROPERTY_ASSERTIONS);
7  hermit.precomputeInferences(InferenceType.CLASS_ASSERTIONS);
```

Listing 14: Konfigurationsparameter für den Inferenztyp des Hermit Reasoner

Listing 14 zeigt einen Ausschnitt der möglichen Hermit Konfigurationen. Über Parameter können verschiedene Inferenztypen zu der Berechnung hinzugefügt werden. Beispiele sind Klassenzugehörigkeiten, Objekt- und Dateneigenschaften und verschiedene Hierarchietypen. Je mehr Konfigurationen miteinbezogen werden, desto länger dauert der Vorgang. Die Inferenzdauer beträgt ca. 30 Sekunden.

7.3.3.2 Abfragedauer

In Diagramm 3 sind die Abfragezeiten sowohl für die Referenzimplementierung als auch für den Tripelstore aufgeführt.

Für den Tripelstore war die Dauer für alle Abfragen etwa gleich lang. Für die Referenzimplementierung waren die Zeiten für die verschiedenen Abfragen wesentlich geringer. Einzig für die Klasse *Plattform* war (wie zu erwarten) eine vergleichsweise hohe Dauer von 36 Millisekunden zu verzeichnen. Dies resultierte aus dem komplexen Aufbau der innerhalb der Klasse notwendigen Komponenten (vgl. Listing 15).

Die Zeiten für den Tripelstore sind ähnlich, da zum einen die SPARQL Abfrage identisch ist, lediglich der Name der Klasse ändert sich. Zum anderen sind die Daten in der Datenbank optimiert angelegt, wodurch ähnliche Zugriffszeiten ermöglicht werden.

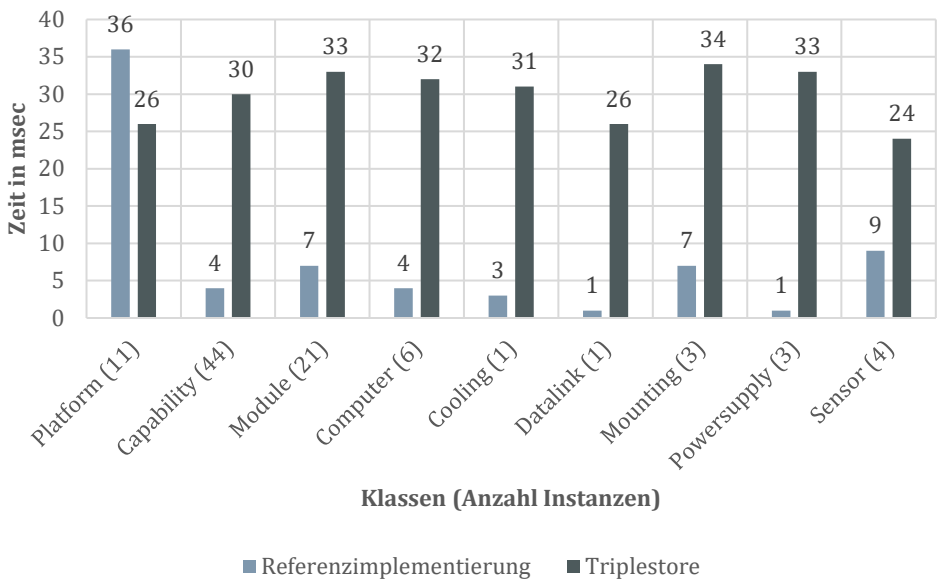


Diagramm 3: Vergleich der Abfragezeiten verschiedener Klassen

Die sehr geringen Abfragezeiten für die Referenzimplementierung kommen deshalb zustande, weil die Datei bereits im Speicher der Softwareapplikation liegt (während die Daten im Tripelstore in einer externen Applikation, der Datenbank liegen). Somit hat man in der Softwareapplikation unmittelbaren Zugriff auf die Daten. Jedoch müssen, nachdem die Daten ausgelesen werden, diese erst miteinander über ihre Beziehungen verknüpft werden. Je mehr Beziehungen überprüft werden müssen, desto länger dauert der Prozess (zu sehen am Beispiel der *Platform*).

In Diagramm 4 wurden durchschnittlichen Abfragezeiten für verschiedene Klassen gemessen. Diese Messungen wurden lediglich für den Tripelstore durchgeführt, da die reinen Abfragezeiten bei der Referenzimplementierung sehr gering waren.

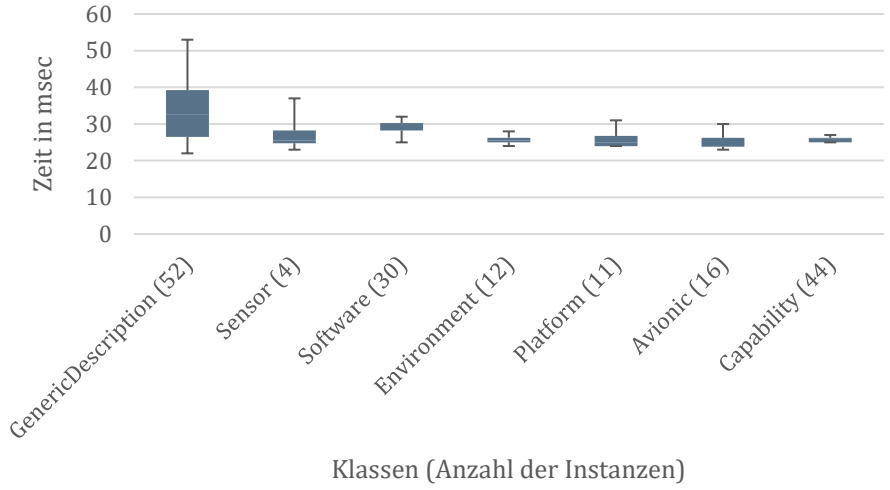


Diagramm 4: Ermittlung von Instanzen einer Klasse

Die Messungen zeigten, dass die Abfragedauer der verschiedenen Klassen unabhängig von der Anzahl der darin enthaltenen Instanzen war. Dies deckt sich mit den Daten in Diagramm 3.

7.3.3.3 Gesamtdauer

Die Gesamtdauer von vier Abfragen für beide Ansätze ist in Diagramm 5 dargestellt.

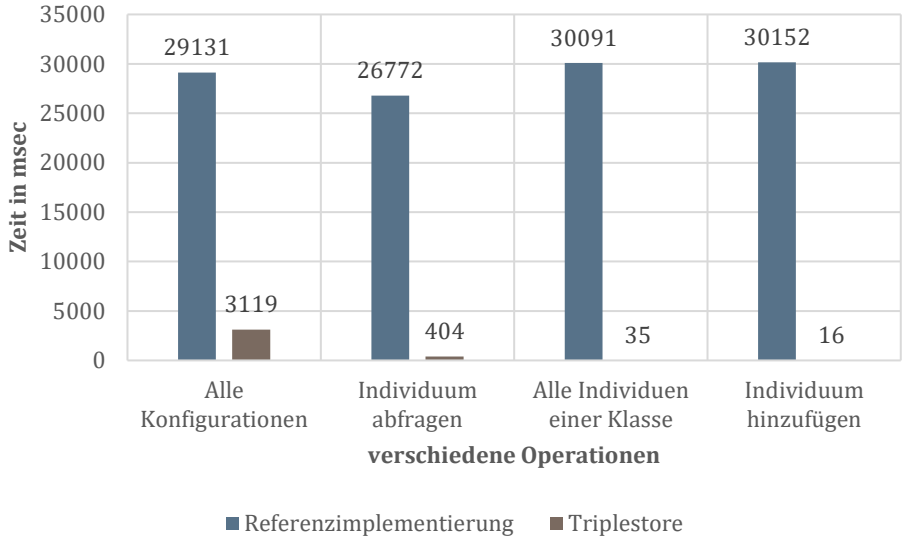


Diagramm 5: Verschiedene Operationen mittels der Referenzimplementierung als auch des Tripelstores

Aus den Angaben in Diagramm 5 wird der Leistungsunterschied deutlich. Für jede Operation muss bei der Referenzimplementierung mit ~30 Sekunden gerechnet werden, während die Abfrage beim Tripelstore im geringfügigen Millisekunden-Bereich liegt.

7.3.4 Fazit der Leistungsbewertung

Die Experimente untersuchten den Zeitbedarf für das Extrahieren von Informationen aus der Ontologie.

Zunächst wurde die Inferenzdauer ermittelt (vgl. Kapitel 7.3.3.1). Es wurde festgestellt, dass die Referenzimplementierung ca. 30 Sekunden benötigt, um alle Inferenzen zu bilden. Der Tripelstore benötigte dafür lediglich Millisekunden im zweistelligen Bereich. Das Ressourcenmodell wurde in einer dynamischen Mission unter der Annahme regelmäßiger System- sowie Umweltveränderungen eingesetzt. Dadurch musste der Inferenzmechanismus in regelmäßigen Abständen verwendet werden. In den Anforderungen wurde festgelegt, dass das Detektieren von Veränderungen und das Reagieren darauf in einem geringen Sekundenbereich liegen sollten. Schlussfolgernd disqualifiziert sich die Referenzimplementierung wegen der langen Inferenzdauer.

Die Untersuchung der Abfragedauer (vgl. Kapitel 7.3.3.2) ergab, dass, sobald die inferierte Ontologie bereitsteht, die Abfragen um ein Vielfaches schneller (im einstelligen Millisekunden Bereich) als die des Tripelstores (zweistelliger Millisekunden Bereich) bearbeitet werden.

Bei den Experimenten zur Ermittlung der Gesamtdauer (vgl. Kapitel 7.3.3.3) stellt sich heraus, dass trotz der geringen Abfragezeiten, die Referenzimplementierung für das im Rahmen dieser Arbeit entwickelte Softwaresystem nicht in Frage kommt.

Die Ergebnisse zeigen jedoch, dass für eine Anwendung, in der die Inferenz z.B. nur bei Systemstart einmalig ausgeführt wird, die Referenzimplementierung verwendet werden kann. Der Vorteil liegt hierbei nicht nur in den schnelleren Abfragezeiten, sondern auch in der geringeren Systemkomplexität, da weder eine Datenbank noch Schnittstellen zur Kommunikation mit dieser benötigt werden. Ein Szenario, in dem die Referenzimplementierung im Rahmen dieser Arbeit verwendet werden konnte, ist die in Kapitel 7.2.1 beschriebene Anwendung, in der die Ontologie zur Missionsvorbereitung genutzt wird. In dieser Anwendung ist nicht mit Veränderungen in der Ontologie (während der Benutzung) zu rechnen, wodurch ein einmaliges Ausführen der Inferenzmaschine ausreicht.

7.4 Bewertung

In den vorhergehenden Kapiteln wurden die Ontologie und das damit verbundene Softwaresystem auf verschiedene Eigenschaften hin untersucht. Im Folgenden soll zunächst eine zusammenfassende Bewertung des entwickelten Systems hinsichtlich der in Kapitel 3 aufgestellten Anforderungen an das Modell durchgeführt und anschließend die Grenzen der vorgenommenen Modellbildung aufgezeigt werden.

7.4.1 Anforderungen an das Modell

Integrierbarkeit in das bestehende Systemkonzept

In Kapitel 4 wurde durch die Erweiterung des S&PM gezeigt, wie das Ressourcenmodell in die bestehende Architektur erfolgreich integriert wurde. Es wurde beschrieben, wie das bereits bestehende S&PM um eine ontologiebasierte Komponente erweitert wurde. In Kapitel 7.2 konnte anschließend das erweiterte S&PM erfolgreich in einer simulierten, operationellen Mission angewandt werden. Die Grundarchitektur des S&PM wurde dabei beibehalten und der Zugriff auf die ontologiebasierte Komponente durch geeignete Schnittstellen gewährleistet. Weiterhin wurde die Konzepterweiterung so vorgenommen, dass sie im Rahmen weiterer Forschungsarbeiten genutzt und um weitere Komponenten erweitert werden konnte (vgl. [61]).

Aussagefähigkeit hinsichtlich möglicher Ressourcenkombinationen

In Kapitel 7.2 wurde anhand von Experimenten demonstriert, dass die Ontologie während der Missionsvorbereitung sowohl zur Ermittlung von UAV-Konfigurationen für eine spezifische Missionsaufgabe, als auch zur Ermittlung von Fähigkeiten unterschiedlicher UAV-Konfigurationen eingesetzt werden kann (*Top-Down* und *Bottom-Up* Ansatz). Es konnte validiert werden, dass mittels der Ontologie perzeptive Fähigkeiten zutreffend ermittelt und die dafür notwendigen Ressourcenkombinationen korrekt identifiziert werden konnten.

Reaktionsfähigkeit bei Zustandsveränderung

Im Rahmen der operationellen Validierung (vgl. Kapitel 7.2) wurde in einem dynamischen Missionsszenario dargestellt, dass das System in der Lage ist, auf Zustandsveränderungen zu reagieren. Durch die Simulation von Hardwareausfall und sich verändernden Umweltbedingungen wurde das Reaktionsverhalten des Systems untersucht. Die Ergebnisse zeigten, dass das System, wie gefordert, auf die verändernden Bedingungen reagierte und der zeitliche Rahmen eingehalten wurde.

Einhaltung begrenzter Verarbeitungszeiten

In Kapitel 7.3 wurden die Antwortzeiten verschiedener Anfragen an die Ontologie untersucht. Anhand der Ergebnisse konnte festgestellt werden, dass nur bei Verwendung der Tripelstore-Implementierung die notwendigen Verarbeitungszeiten, wie gefordert, eingehalten wurden. Für eine Anwendung am Boden, bei der Verarbeitungszeiten unkritisch sind, kommt durchaus auch die OWL Referenzimplementierung für einen operationellen Einsatz in Frage.

Domänenübergreifende Konzeptrepräsentation

Durch die in Kapitel 6.2.1 vorgestellte Integration von Ontologien Dritter wurde gezeigt, dass die im Rahmen dieser Arbeit entwickelte Ontologie in einem ganzheitlichen Kontext außerhalb der Domäne betrachtet werden kann. Durch die Integration der Top-Level Ontologie DUL wurde demonstriert, wie Konzepte in Relation zu anderen Konzepten außerhalb der Domäne hergestellt werden können. Durch die strukturelle Bewertung in Kapitel 7.1 wurde nachgewiesen, dass allgemein gültige Designkriterien und *best practices* soweit wie möglich eingehalten wurden. Durch den modularen Aufbau der Ontologie (vgl. Kapitel 6.2.2) können bereits bestehende Konzepte ausgetauscht, erweitert oder getrennt von anderen Konzepten wiederverwendet werden.

7.4.2 Grenzen des Modells

In Anbetracht der Aufgabenstellung und der damit verbundenen Anforderungen konnte im Rahmen der Validierung gezeigt werden, dass das zu entwickelnde Konzept und die holistische Ressourcenbeschreibung den Anforderungen genügt und somit für den Einsatz von luftgestützten Aufklärungsaufgaben genutzt werden kann. Allerdings wurden während der Entwicklung und Validierung der Ontologie die Grenzen des Modells und Modellierungsungenauigkeiten festgestellt. Diese sollen im Folgenden näher erläutert werden.

Interoperabilität

Wie bereits aus den Erläuterungen in Kapitel 5.3.3 ersichtlich, gestaltet es sich schwierig, passende existierende Ontologien für eine bestimmte Aufgabe zu finden. Ontologien wurden oft für ein spezifisches, domänenabhängiges Problem entwickelt, wodurch es selten möglich ist, eine vollständige Ontologie Dritter zu übernehmen. Aus diesem Grund wurde im Rahmen dieser Arbeit eine neue Ontologie entwickelt. Die modellierten Konzepte wurden hinreichend genug beschrieben, damit das Ressourcenmodell für luftgestützte Aufklärungsmissionen genutzt werden kann. In Zukunft wäre es denkbar, die vorhandenen Konzepte präziser und mit mehr

Detailtiefe zu beschreiben. Konzepte anderer Ontologien (z.B. Sensoren oder die ATA-Kapitel) wurden in die zu entwickelnde Ontologie importiert, um Verknüpfungen zu anderen Ontologien herzustellen. Durch Verwendung der Top-Level Ontology DUL wurde die Ontologie in einen allgemeinen Kontext außerhalb der Luftfahrtdomäne eingegliedert.

Komplexe Berechnungen

Eine Ontologie ist geeignet, generelle Beziehungen zwischen Konzepten zu beschreiben. Diese können genutzt werden, um Konzepte, Zusammenhänge und Abhängigkeiten einer Domäne zu verstehen. Mittels *Dataproperties* können ebenfalls konkrete Daten mit Konzepten verknüpft werden. Weiterhin besteht die Möglichkeit, für Daten und Beziehungen Restriktionen festzulegen, z.B. dass das Gewicht einer Sensornutzlast einen bestimmten Wert nicht überschreiten darf. Eine Ontologie stößt jedoch an ihre Grenzen, sobald komplexe mathematische Berechnungen durchgeführt werden müssen. Beispielsweise kann die Reichweite eines Luftfahrzeugs innerhalb der Ontologie modelliert werden, jedoch ist diese nicht geeignet, um z.B. während einer Mission die Reichweite der Plattform in Abhängigkeit von Treibstoffverbrauch, Luftwiederstand und anderen flugtechnischen Parameter zu ermitteln. Die Informationen, die zur Berechnung der Reichweite notwendig sind, und von welchen Parametern diese abhängen, könnten zwar in der Ontologie modelliert werden (in Form von Beziehungen), die Berechnung selbst muss jedoch separat durchgeführt werden.

Umfang und Genauigkeit des Modells

Bei der operationellen Validierung in Kapitel 7.2 wurde gezeigt, dass das Modell genutzt werden kann, um UAV-Konfigurationen zu identifizieren, die geeignet sind, spezifische Aufklärungsaufgaben durchzuführen. Hierfür wurde angenommen, dass alle vorhandenen Komponenten Plug & Play fähig sind und somit untereinander ausgetauscht werden können. Diese Annahme entspricht, trotz fortschreitender Standardisierung, nicht dem aktuellen Stand der Technik.

Zudem wurde die Anzahl der verwendeten Beschreibungsparameter der einzelnen Ressourcenelemente bewusst gering gehalten, sodass nur solche modelliert wurden, die für die Aufgabenstellung notwendig waren. Neben den beschriebenen Parametern existieren viele weitere Parameter für eine Ressource, die im Rahmen dieser Arbeit nicht modelliert wurden. Auch nutzt das Modell einen stark vereinfachten Ansatz, um die Integration von Ressourcen in ein UAV zu modellieren. Beispielsweise wird für einen Gimbal lediglich festgelegt, dass er in einem UAV integriert wurde, an welcher Stelle der Gimbal jedoch befestigt werden kann, wurde in der Ontologie nicht abgebildet. Dies gilt es zu erweitern.

8

Ausblick

Im Rahmen dieser Forschungsarbeit wurde ein ontologiebasiertes Ressourcenmanagementsystem implementiert. Ziel war es ein holistisches, maschinenverarbeitbares Modell zur Beschreibung von Ressourcen und Fähigkeiten bereitzustellen. In Kapitel 7 wurden in verschiedenen Experimenten gezeigt, dass das Ressourcenmodell den Anforderungen aus Kapitel 3 genügt und die im Rahmen dieser Arbeit gestellte Aufgabenstellung mit dem Modell gelöst werden konnte. Die Experimente haben aber auch gezeigt, dass Potential besteht, das Ressourcenmodell zu erweitern oder anderweitig zu nutzen. Im Folgenden sollen einige weitere Szenarien vorgestellt werden, die für den Einsatz dieser Ontologie in Frage kommen.

Wie bereits kurz in Kapitel 7.2.2.3 angesprochen, liefert das Ressourcenmodell in Abhängigkeit der Ausstattung des Luftfahrzeugs und der durchzuführenden Aufgabe mehrere Lösungen, um eine Aufgabe auszuführen. In [61] wird die im Rahmen dieser Arbeit entwickelten Ontologie soweit erweitert, dass eine Bewertung der verschiedenen Lösungen mittels unterschiedlicher Metriken stattfinden kann. Abhängig von Umwelt, Systemzustand und Aufgabe kann so entschieden werden, welche Lösungsstrategie die größte Erfolgswahrscheinlichkeit hat.

Im Rahmen dieser Arbeit wurde die Ontologie (vgl. Kapitel 7.2.2) mit Fokus auf eine ein-Operateur-ein-UAV-Konfiguration entwickelt. Denkbar ist der Einsatz des S&PM in einer Multi-UAV-Konfiguration in der jedes UAV ebenfalls als Ressource betrachtet wird. Die UAVs unterscheiden sich dabei in ihrer eigenen Ausprägung (Flugeigenschaften, Größe, Geschwindigkeit, etc.) als auch in der Sensorausstattung und mitgeführten Bildverarbeitungsalgorithmen respektive Perzeptionsmodule. Eine mögliche Untersuchung der Ontologie könnte darauf abzielen, wie ein ontologiebasierter Ansatz einen Mehrwert in einer Multi-UAV Umgebung mit sich bringt. Dabei stehen zwei Optionen zur Auswahl. Zum einen besteht die Möglichkeit, eine Ontologie in einem zentralen System zu verwalten. Jedes UAV übermittelt seine Fähigkeiten an die zentrale Ontologie und diese identifiziert mittels maschinellen Schließens, welches UAV welche Aufgaben durchführen soll. Zum anderen sieht die zweite Option es vor, dass jedes UAV seine eigene „kleine“ Ontologie mit sich führt und jedes der UAVs für sich selbst ermittelt, welche Fähigkeiten zu einer bestimmten Situation zur Verfügung stehen.

In Kapitel 7.2.1 wurde darauf hingewiesen, dass im Rahmen des Experiments angenommen wird, dass die Ressourcen Plug & Play-fähig sind, d.h. verschiedene Ressourcen miteinander kombiniert werden können. In der Praxis ist dieser Fall die Ausnahme. In Zukunft wäre es jedoch denkbar, dass jede Ressource ihre eigene Ontologie mit sich führt. Soll nun ermittelt werden, ob und wie bestimmte Ressourcen miteinander kombiniert werden können und welche Fähigkeiten sich aus dessen Kombination ergebenen, könnte man einzelnen Ontologien der Ressourcen zusammenführen und anschließend über eine Inferenzmaschine ermitteln, in welchem Zusammenhang die Ressourcen miteinander stehen.

Zuletzt wurden in Kapitel 7.4.2 die Grenzen des Models erörtert. Dabei wurde festgestellt, dass die im Rahmen dieser Arbeit entwickelte Ontologie zwar ausreicht, um die Anforderungen zu erfüllen, jedoch Modellierungsungenauigkeiten in der Ontologie existieren. Soll diese Ontologie in einem anderen Aufgabenfeld genutzt werden, kann es notwendig sein, je nach Komplexität und Problemstellung, die Ontologie zu erweitern, indem weitere Konzepte hinzugefügt, Ontologien Dritter inkludiert oder Modellierungsungenauigkeiten durch spezifischere Beschreibungen der bestehenden Konzepte beseitigt werden.

Quellenverzeichnis

[1] D. Hausamann et al., "Monitoring of gas pipelines – A civil UAV application," *Aircraft Engineering and Aerospace Technology*, vol. 77, no. 5, 2005, pp. 352–360.

[2] T. R. Bretschneider and K. Shetti, "UAV-based gas pipeline leak detection," in *Proceedings of Asian Conference on Remote Sensing*, 2015.

[3] S. M. Adams and C. J. Friedland, "A Survey of Unmanned Aerial Vehicle (UAV) Usage for Imagery Collection in Disaster Research and Management," in *Proceedings of the 9th International Workshop Remote Sensing for Disaster Response*, Stanford, CA, USA, 2011.

[4] C. Zhang and J. M. Kovacs, "The application of small unmanned aerial systems for precision agriculture: A review," *Precision Agriculture*, Springer, vol. 13, no. 6, 2012, pp. 693–712.

[5] G. Grenzdörffer, A. Engel and B. Teichert, "The photogrammetric potential of low-cost UAVs in forestry and agriculture", *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences,* Beijing, 2008. pp. 1207–1213.

[6] S. Candiago et al., "Evaluating multispectral images and vegetation indices for precision farming applications from UAV images," *Remote Sensing*, vol. 7, no. 4, 2015, pp. 4026–4047.

[7] M. Russ et al., "Airborne sensor and perception management: Experiments and Results for surveillance UAS," in *AIAA Infotech@ Aerospace (I@A) Conference*, 2013, p. 5144.

[8] G. R. Rodriguez-Canosa et al., "A real-time method to detect and track moving objects (DATMO) from unmanned aerial vehicles (UAVs) using a single camera," *Remote Sensing*, vol. 4, no. 4, 2012, pp. 1090–1111.

[9] K. L. B. Cook, "The Silent Force Multiplier: The History and Role of UAVs in Warfare," in *Proceedings of 2007 IEEE Aerospace Conference*, 2007, pp. 1-7.

[10] J. D. Blom, "Unmanned Aerial Systems: A Historical Perspective," vol. 45, Combat Studies Institute Press, KS, USA, 2010.

[11] R. Schaefer, "Unmanned Aerial Vehicle Reliability Study," Office of the Secretary of Defense, Washington, DC, 2003, pp. 4–5.

[12] R. Kumar, "Tactical Reconnaissance: UAVs versus Manned Aircraft," Research Paper, The Research Department Air Command and Staff College, No. AU/ACSC/0349/97-03, Maxwell Air Force base, Alabama 1997.

[13] K. Farid, "Survey of Advances in Guidance, Navigation, and Control of Unmanned Rotorcraft Systems," *Journal of Field Robotics*, vol. 29, no. 2, 2011, pp. 315–378.

[14] R. Austin, *Unmanned Aircraft Systems: UAVS design, development and deployment*, John Wiley & Sons Ltd, United Kingdom, 2011.

[15] M. J. Barnes et al., "Crew systems analysis of unmanned aerial vehicle (UAV) future

job and tasking environments," Rep. TR2081, U.S. Army Research Labaratory, Adelphi, MD, 2000.

[16] M. J. Barnes and M. F. Matz, "Crew Simulations for Unmanned Aerial Vehicle (UAV) Applications: Sustained Effects, Shift Factors, Interface Issues, and Crew Size," in *Proceedings of Human Factors Ergonomics Society Annual Meeting*, vol. 42, no. 1, Sage CA, Los Angeles, 1998, pp. 143–147.

[17] B. Walters, J. French and M. J. Barnes, "Modeling the effects of crew size and crew fatigue on the control of tactical unmanned aerial vehicles (TUAVs)," in *Proceedings of Simulation Conference*, vol. 1, 2000, pp. 920-924.

[18] M. L. Cummings et al., "Automation Architecture for Single Operator-Multiple UAV Command and Control," *International C2 Journal*, vol. 1, no. 2, 2007, pp. 1–24.

[19] J. Uhrmann, R. Strenzke and A. Schulte, "Task-based Guidance of Multiple Detached Unmanned Sensor Platforms in Military Helicopter Operations," *COGnitive systems with Interactive Sensors*, Crawley UK, 2010.

[20] Naval Surface Warfare Center-Dahlgren Division, "Tactical Control System (TCS) System / Subsystem Specification," Cruise Missiles Project and Unmanned Air Vehicles Join Project, February, 1999. [Online]. Available: https://de.scribd.com/document/ 229330722/139468451-Tactical-Control-System-TCS-System-Subsystem-Specification. [Accessed: 19-Jan-2016]

[21] A. Rauschert and A. Schulte, "Cognitive and cooperative assistant system for aerial manned-unmanned teaming missions," *Supervisory Control of Multiple Uninhabited Systems – Methodologies and Enabling Human-Robot Interface Technologies (NATO RTO Task Group HFM-170 Report)*, vol. 170, 2012, pp. 1–16.

[22] G. Rudnick, S. Clauß and A. Schulte, "Flight Testing of Agent Supervisory Control on Heterogenous Unmanned Aerial System Platforms," in *Proceedings of Digital Avionics Systems Conference*, 2014, pp. 4B2-1.

[23] N. Theißing and A. Schulte, "Flight Management Assistance Through Cognitive Automation Adapting to the Pilot' s State of Mind Mentalizing Pilot Assistance," *31st Conference European Assocciation for Aviation Psychology*, 2014.

[24] K. W. Williams, "A Summary of Unmanned Aircraft Accident/Incident Data: Human Factors Implications," U.S. Department of Transportation, Federal Aviation Administration, Civil Aerospace Medical Institute, Oklahoma City, vol. 4, 2004, pp. 1–5.

[25] United States Air Force, "United States Air Force Accident Investigation Board Reports," 2012. [Online]. Available: http://usaf.aib.law.af.mil/. [Accessed: 19-Jan-2016].

[26] J. S. McCarley and C. D. Wickens, "Human factors implications of UAVs in the national airspace," Tech. Rep. AHFD-05-05/FAA-05-01, Institute of Aviation, Aviation Human Factors Division, University of Illinois at Urbana-Champaign, Savoy, IL, 2005.

[27] S. Gangl, B. Lettl and A. Schulte, "Single-seat Cockpit-based Management of Multiple UCAVs Using On-board Cognitive Agents for Coordination in Manned-unmanned Fighter Missions," in *International Conference on Engineering Psychology and Cognitive Ergonomics. Applications and Services*, Springer, Berlin, 2013, pp. 115–124.

[28] L. L. Coulter et al., "Near Real-Time Change Detection for Border Monitoring," in

Proceedings from the ASPRS annual conference, Milwaukee, USA, 2011, pp. 1–5

[29] G. Hummel et al., "Prototyping and training of computer vision algorithms in a synthetic UAV mission test bed," in *52nd AIAA SciTech*, 2014, p. 1322.

[30] G. Hummel et al., "Intelligent Multi Sensor Fusion System for Advanced Situation Awareness in Urban Environments," in *Future Security*, vol. 318, Springer, Berlin, 2012, pp. 93–104.

[31] P. Campoy et al., "Computer vision onboard UAVs for civilian tasks," *Journal of Intelligent Robotic Systems*, vol. 54, no. 1–3, 2008, pp. 105–135.

[32] R. Stevens et al., "Improved Aerial Surveillance Using Onboard Image Processing," *SPIE Newsroom Article 27*, October, 2008.

[33] C. P. C. Chanel, F. Teichteil-Königsbuch and C. Lesire, "Multi-target detection and recognition by UAVs using online POMDPs," in *Proceedings of 27th AAAI Conference for Artificial Intelligence*, July, 2013, pp. 1381–1387.

[34] S. Musick and R. Malhotra, "Chasing the Elusive Sensor Manager," in *IEEE National Aerospace and Electronics Conference*, vol. 1, 1994, pp. 606–613.

[35] F. Smits et al., "A Cognitive Radar Network : Architecture and Application to Multiplatform Radar Management," in *IEEE Radar Conference, 2008,* pp. 312-315.

[36] S. Miranda et al., "Knowledge-Based Resource Management for Multifunction Radar," in *IEEE Signal Processing Magazine*, January, 2006, pp. 66–76.

[37] P. Skoglar, "UAV Path and Sensor Planning Methods for Multiple Ground Target Search and Tracking - A Literature Survey," Department of Electrical Engineering, Linköping University Electronic Press, SE-581 83 Linköping, Sweden, 2007.

[38] P. Skoglar, J. Nygards and M. Ulvklo, "Concurrent Path and Sensor Planning for a UAV - Towards an Information Based Approach Incorporating Models of Environment and Sensor," in *IEEE International Conference on Intelligent Robots and Systems*, 2006, pp. 2436–2442.

[39] M. Russ and P. Stütz, "Airborne sensor and perception management: A conceptual approach for surveillance UAS," in *15th International Conference for Information Fusion*, 2012, pp. 2444–2451.

[40] D. Oberle et al. "DOLCE ergo SUMO: On foundational and domain models in the SmartWeb Integrated Ontology (SWIntO)," *Web Semantics: Science, Services and Agents on the World Wide Web* 5.3, 2007, pp. 156-174.

[41] P. Hitzler et al., *Semantic Web*, Springer, Berlin, 2007.

[42] C. Hellert et al., "A High Level Active Perception Concept For UAV Mission Scenarios," in *Deutscher Luft- und Raumfahrtkongress*, 2012.

[43] D. Smirnov and P. Stütz, "Knowledge elicitation and representation for module based perceptual capabilities onboard UAVs," in *52nd Aerospace Sciences Meeting*, 2014, p. 1321.

[44] D. Smirnov and P. Stütz, "Ontology-Based Modelling of Sensor and Data Processing Ressources Using OWL: A Proof of Concept," in *The First International Conference on Applications and Systems of Visual Paradigms*, 2016.

[45] D. Smirnov and P. Stütz, "Use case driven approach for ontology-based modeling of reconnaissance resources on-board UAVs using OWL," in *IEEE Aerospace*

Conference, 2017, pp.1-17.

[46] C. Hellert, D. Smirnov and P. Stütz, "Ontologiedesign für Sensor- und Perzeptionsfähigkeiten von UAVs," in *Deutscher Luft- und Raumfahrtkongress*, 2014.

[47] R. Szeliski, *Computer Vision - Algorithms and Applications*, Springer, London, 2011.

[48] D. A. Forsyth and J. Ponce, *Computer Vision - A Modern Approach*, 2nd Edition, Pearson Education Limited, 2012.

[49] B. Jähne, *Digitale Bildverarbeitung*, Springer, Berlin, 2012.

[50] World Trade Organization, "Natural resources : Definitions, Trade Patterns and Globalization," *World Trade Report*, 20110, pp. 44–71.

[51] G. Bohlander and S. Snell, *Managing Human Resources*, 14th Edition, Mason, USA, 2007.

[52] Duden, "Bedeutung: Ressource," 2016. [Online]. Available: http://www.duden.de/rechtschreibung/Ressource#Bedeutung1. [Accessed: 21-Jan-2016].

[53] E. B. Worthington, "A definition of natural resources," in *Conference on the organization of research and training in Africa in relation to the study, conservation and utilization of natural resources*, 1964, pp. 1–11.

[54] G. Steiner et al., "Analysis of the Fundamental Concepts of Resource Management," Gesellschaft für umfassende Analysen GmbH, Vienna, 1996.

[55] L. Halawi, J. Aronson and R. McCarthy, "Resource-based view of knowledge management for competitive advantage," *The Electronic Journal of Knowledge Management*, vol. 3, no. 2, 2005, pp. 75–86.

[56] Jackson S., *Systems engineering for commercial aircraft*, Routledge, London, 2016.

[57] M. Shekarriz, A. Mousavi and P. Broomhead, "On the definition of capability in the workplace, a new perspective – part 1," *Engineering Management*, 2013.

[58] J. B. Barney, "How a Firm's Capabilities Affect Boundary Decisions," *Sloan Management Review*, vol. 40, no. 3, 1999, p. 137.

[59] Duden, "Bedeutung: Fähigkeit," 2016. [Online]. Available: http://www.duden.de/rechtschreibung/Faehigkeit. [Accessed: 21-Jan-2016].

[60] T. Schuster, "Modellierung, Integration und Analyse von Ressourcen in Geschäftsprozessen," KIT Scientific Publishing, 2012.

[61] C. Hellert and P. Stütz, "Performance Prediction and Selection of Aerial Perception Functions during UAV Missions," in *IEEE Aerospace Conference*, 2017, pp. 1-12.

[62] P. Morrison, "White Paper: VBS2 Release Version 2.0," Nelson Bay, Australia, 2012.

[63] N. Guarino and C. Welty, "An overview of OntoClean," in *Handbook on ontologies,* Springer, Berlin, 2004, pp. 151–171.

[64] K. Kurbel, *Entwicklung und Einsatz von Expertensystemen*, Springer, Berlin, 1992.

[65] P. Norvig and S. Russel, *Künstliche Intelligenz: Ein moderner Ansatz*, 2nd Edition, Pearson Studium, Munich, 2004.

[66] G. F. Luger, *Künstliche Intelligenz - Strategien zur Lösung komplexer Probleme*, 1st Edition, Pearson Studium, Munich, 2001.

[67] VDI-Gesellschaft Entwicklung Konstuktion Vertrieb, *Wissensbasierte Systeme für Konstruktion und Arbeitsplanung*, VDI Verlag , Düsseldorf, 1992.

[68] D. Altenkrüger and W. Büttner, *Wissensbasierte Systeme: Architektur, Entwicklung, Echtzeitanwendungen - Eine praxisgerechte Einführung*, Vieweg+Teubner Verlag, Wiesbaden, 1992.

[69] U. Reimer, *Einführung in die Wissensrepräsentation*, B. G. Teubner, Stuttgart, 1991.

[70] F. Puppe, *Einführung in Expertensysteme*, 2. Ausgabe, Springer, Karlsruhe, 1991.

[71] A. Dengel, *Semantische Technologien*, Springer Spektrum, Heidelberg, 2012.

[72] B. Motik, R. Shearer and I. Horrocks, "Hypertableau reasoning for description logics," *Journal of Artificial Intelligence Research*, 2009, pp. 165–228.

[73] S. Russell and P. Norvig, *Künstliche Intelligenz*, 3. Ausgabe, Pearson Deutschland, Bremen, 2012.

[74] P. Besnard, *An Introduction to Default Logic*, Springer, Berlin, 1989.

[75] B. Becker, *Künstliche Intelligenz: Konzepte, Systeme, Verheissungen*, Campus Verlag, Frankfurt, 1992.

[76] M. Quillian, "Semantic memory," in M. Minsky (ed.) *Semantic Information Processing*, Cambridge, MA: The MIT Press, 1968.

[77] K. H. Wagner, "Darstellung von Wissen," pp. 1–7, 2003. [Online]. Available: http://www.fb10.uni-bremen.de/khwagner/semantik/pdf/wissen.pdf. [Accessed: 21-Jan-2016].

[78] F. Baader et al., *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press, *2003*.

[79] G. De Giacomo and M. Lenzerini, "TBox and ABox Reasoning in Expressive Description Logics," in *Proceedings of 5th International Conference on the Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, Los Altos, 1996, pp. 316-327.

[80] H. Stuckenschmidt, *Ontologien*. Springer, Heidelberg, 2010.

[81] M. P. Villalón and A. G. Pérez, *Ontology Evaluation: a pitfall-based approach to ontology diagnosis*, Doctoral Dissertation thesis, Universidad Politécnica de Madrid, 2016.

[82] S. Staab and C. Brewster, "Knowledge Representation with Ontologies : The Present and Future," *IEEE Intelligent Systems*, vol.19, no.1, 2004, pp. 72–81.

[83] F. J. Furrer, "Eine kurze Geschichte der Ontologie," *Informatik-Spektrum*, vol. 37, 2014, pp. 308–317.

[84] J. F. Sowa, http://www.jfsowa.com/ontology/, [Online], [Accessed: 21-Jan-2016].

[85] P. D. Karp, "An ontology for biological function based on molecular interactions," *Bioinformatics*, vol. 16, 2000, pp. 269–85.

[86] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, 1993, pp. 199–220.

[87] M. Uschold and M. Gruninger, "Ontologies : Principles , Methods and Applications," *The Knowledge Engineering Review*, vol. 11, 1996, pp. 93-136.

[88] W. G. Stock and M. Stock, *Wissensrepräsentation: Informationen auswerten und*

bereitstellen, Oldenbourg Verlag, Munich, 2008.

[89] A. Fenk, "Semiotische Dreiecke als Problem für den radikalen Konstruktivismus," *in* V. A. Munz, K. Puhl, J. Wang (eds.) *Papers from the International Wittgenstein Symposia* ,Kirchberg am Wechsel, 2014.

[90] C. K. Ogden and I. A. Richards, *The Meaning of Meaning*. Brace & Co, New York, 1923.

[91] N. F. Noy and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," *Stanford Knowledge System Lab*, 2001.

[92] E. Simperl et al. (eds.), *The Semantic Web: Research and Applications: 9th Extended Semantic Web Conference,* Springer, Crete, Greece, *v*ol. 7295, 2012.

[93] N. Guarino (ed.), *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98),* IOS press, Trento, Italy, vol. 46., 1998.

[94] O. Lassila and D. L. McGuinness, "The Role of Frame-Based Representation on the Semantic Web," *Linköping Electronic Articles in Computer and Information Science*, vol. 12, 2001.

[95] J. H. Gennari et al., "The evolution of Protégé: An Environment for Knowledge-based Systems Development," *International Journal of Human-computer studies*, vol. 58, 2003, pp. 89–123.

[96] A. Jürgen and Y. Sure. "Evaluation of ontology-based Tools," in *Proceedings of the OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management EKAW*, CEUR Workshop Proceedings, vol. 62, 2002.

[97] M. Kroetzsch, "OWL 2 profiles: An introduction to lightweight ontology languages," in *Reasoning Web International Summer School,* Springer, Berlin, 2012, (pp. 112-183).

[98] W3C Recommendation, "W3C OWL 2," 2012. [Online]. Available: https://www.w3.org/TR/owl2-overview/. [Accessed: 10-Jul-2017].

[99] P. Hitzler, M. Krötzsch and S. Rudolph, *Foundations of Semantic Web Technologies,* Chapman & Hall/CRC, Boca Raton, USA, 2009.

[100] C. Martínez-Costa and S. Schulz, "Ontology-based reinterpretation of the SNOMED CT context model," in *CEUR Workshop Proceedings*, vol. 1060, 2013, pp. 90–95.

[101] H.-D. Ebbinghaus, J. Flum and W. Thomas, *Einführung in die mathematische Logik*, 5th Edition, Springer Spektrum, Berlin, 2007.

[102] I. Horrocks et al., "SWRL : A Semantic Web Rule Language Combining OWL and RuleML," in *W3C Member Submission 21*, 2004, pp. 1–20.

[103] A. Gangemi, "Ontology design patterns for semantic web content," in *International Semantic Web Conference*, 2005, pp. 262–276.

[104] M. Gomez et al., "An ontology-centric approach to sensor-mission assignment," in *International Conference on Knowledge Engineering and Knowledge Management*, 2008, pp. 347–363.

[105] A. Preece et al., "An ontology-based approach to sensor-mission assignment," in *1st Annual Conference of the International Technology Alliance*, 2007.

[106] D. Russomanno, C. Kothari and O. Thomas, "Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models," in *The 2005 International Conference*

on Artificial Intelligence, Las Vegas, NV, USA, 2005, pp. 637-643.

[107] T. Devadithya et al. , "The common instrument middleware architecture: Overview of goals and implementation," in *Proceedings of the First IEEE International Conference on e-Science and Grid Computing*, 2005, pp. 578–585.

[108] L. Bermudez, J. Graybeal and R. Arko, "A marine platforms ontology: Experiences and lessons," in *Proceedings of the ISWC 2006 Workshop on Semantic Sensor Networks, Athens GA, USA,* 2006.

[109] D. Schober, V. Svátek and M. Boeker, "Checking class labels against naming conventions: First experience with the OntoCheck Protégé plugin," in *Proceedings of the International Conference on Biomedical Ontology*, 2012.

[110] M. Compton et al., "The SSN ontology of the W3C semantic sensor network incubator group," *Web semantics: science, services and agents on the World Wide Web* 17, 2012, pp. 25–32.

[111] A. Jungmann and B. Kleinjohann, "Automatic Composition of Service-based Image Processing Applications," in *IEEE International Conference on Services Computing (SCC)*, 2016, pp. 106-113.

[112] R. Clouard and A. Elmoataz, "Borg: A knowledge-based system for automatic generation of image processing programs," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 21, no.2, 1999, pp. 128-144.

[113] R. Clouard, A. Renouf and M. Revenu, "An Ontology-Based Model for Representing Image Processing Application Objectives," *International Journal of Pattern Recognition and Artificial Intelligence*, 2010, pp. 1181–1208.

[114] R. Clouard, A. Renouf and M. Revenu, "Human–computer interaction for the generation of image processing applications," *International journal of human-computer studies*, 2011, pp. 201–219.

[115] G. H. Lim, I. H. Suh and H. Suh, "Ontology-Based Unified Robot Knowledge for Service Robots in Indoor Environments," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no.3, 2011, pp. 492–509.

[116] W3C, "Semantic Sensor Network Ontology," 2017. [Online]. Available: https://www.w3.org/TR/vocab-ssn/. [Accessed: 08-Feb-2017].

[117] W3C, "DUL SSN." [Online]. Available: https://www.w3.org/2005/Incubator/ssn/wiki/DUL_ssn. [Accessed: 08-Sep-2017].

[118] W3C, "SSN Skeleton." [Online]. Available: https://www.w3.org/2005/Incubator/ssn/wiki/SSN_Skeleton. [Accessed: 08-Sep-2017].

[119] C. Hellert and P. Stütz, "Selecting Adequate Aerial Perceptual Functions with Fuzzy Logic," in *The First International Conference on Applications and Systems of Visual Paradigms*, 2016.

[120] O. Can et al., "Comparing relational and ontological Tripel stores in Healthcare Domain," *Entropy*, vol. 19, no. 1, 2017, p. 30.

[121] V. Kilintzis, N. Beredimas and I. Chouvarda, "Evaluation of the performance of open-source RDBMS and Tripelstores for storing medical data over a web service," *Engi-*

neering in Medicine and Biology Society (EMBC), 36th Annual International Conference of the IEEE, 2014, pp. 4499–4502.

[122] C. Martinez-Cruz, I. J. Blanco and M. A. Vila, "Ontologies versus relational databases: Are they so different? A comparison," *Artificial Intelligence Review*, 2012, pp. 271–290.

[123] P. Bednar, M. Sarnovsky and V. Demko, "RDF vs. NoSQL databases for the semantic web applications," in *Proceedings of SAMI 2014 - IEEE 12th International Symposium on Applied Machine Intelligence and Informatics,* 2014, pp. 361–364.

[124] P. Cudré-Mauroux et al., "NoSQL databases for RDF: An empirical evaluation," in *International Semantic Web Conference*, Springer, Berlin, 2013, pp. 310–325.

[125] Stardog, "Stardog Documentation." [Online]. Available: http://www.stardog.com/docs/. [Accessed: 08-Sep-2017].

[126] Ó. Corcho et al., "ODEval: A Tool for Evaluating RDF(S), DAML+OIL, and OWL Concept Taxonomies," in *Artificial Intelligence Applications and Innovations: IFIP 18th World Computer Congress TC12 First International Conference on Artificial Intelligence Applications and Innovations (AIAI-2004),* Toulouse, France, Springer, 2004, pp. 369–382.

[127] DB-Engines, "Database Comparison." [Online]. Available: https://db-engines.com/en/system/Neo4j%3BArangoDB%3BMarkLogic%3BOrientDB%3BStardog. [Accessed: 08-Sep-2017].

[128] M. Morsey et al., "Usage-Centric Benchmarking of RDF Tripel Stores," in *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, vol. 26, 2012, pp. 2134–2140.

[129] M. Voigt, A. Mitschick and J. Schulz, "Yet another Tripel store benchmark? Practical experiences with real-world data," in *SDA CEUR Workshop Proceedings*, vol. 912, 2012, pp. 85–94.

[130] W3C Working Group, "OWL 2 Direct Semantics Entailment Regime," 2010. [Online]. Available: https://www.w3.org/TR/2010/WD-sparql11-entailment-20100126/#id45013. [Accessed: 24-May-2017].

[131] M. Horridge and S. Bechhofer, "The OWL API: A java API for OWL Ontologies," *Semantic Web*, 2011, pp. 11–21.

[132] D. Tsarkov and I. Horrocks, "FaCT++ Description Logic Reasoner: System Description," in *International Joint Conference on Automated Reasoning*. Springer, Berlin, 2006, pp. 292–297.

[133] B. Glimm et al., "HermiT: An OWL 2 Reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, 2014, pp. 245–269.

[134] E. Sirin et al., "Pellet: A practical OWL-DL reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, 2007, pp. 51–53.

[135] V. Haarslev and R. Möller, "Racer: An OWL Reasoning Agent for the Semantic Web," in *Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, 2003,* pp. 91–95.

[136] "Ontology Summit 2013," 2013. [Online]. Available: http://ontologforum.org/OntologySummit/2013/. [Accessed: 24-May-2017].

[137] F. Neuhaus et al., "Towards Ontology Evaluation across the Life Cycle," *Applied Ontology,* vol. 8, no.3, 2013, pp. 179–194.

[138] N. Guarino and C. Welty, “Evaluating Ontological Decisions with OntoClean,” *Communications of the ACM*, 2002, pp. 61–65.

Anhang

A. Datenbank Abfragen für 7.3.2.2:

- Platform

```
1  SELECT ?x WHERE { ?x rdf:type ap:Platform}

2  queryDataproperty(?x);
3  queryObjectproperty(?x, "canIntegrateComputer");
4  queryObjectproperty(?x, "canIntegrateMounting");
5  queryObjectproperty(?x, "hasEngineType");
6  queryObjectproperty(?x, "mustIntegrate");
7  queryObjectproperty(?x, "canIntegratePowersupply");
8  queryObjectproperty(?x, "canIntegrateDatalink");
9  queryObjectproperty(?x, "providesEndurance");
10 queryObjectproperty(?x, "providesAltitude ");
11 queryObjectproperty(?x, "providesRange");
12 queryObjectproperty(?x, "providesCapability");
```

Listing 15: SPARQL Abfrage für „Platform“

- Capability

```
1  SELECT ?x WHERE { ?x rdf:type ap:Capability}

2  queryDataproperty(?x);
3  queryObjectproperty(?x, "providesCapability");
```

Listing 16: SPARQL Abfrage für „Capability“

- Module

```
1  SELECT ?x WHERE { ?x rdf:type ap:Module}

2  queryDataproperty(p);
3  queryObjectproperty(p, "canUseModule");
4  queryObjectproperty(p, "requiresModuleOutput");
5  queryObjectproperty(p, "providesModuleOutput");
6  queryObjectproperty(p, "providesMethod");
7  queryObjectproperty(p, "providesEntity");
8  queryObjectproperty(p, "runsOn");
9  queryObjectproperty(p, "providesCapability");
```

Listing 17: SPARQL Abfrage für „Module“

- Computer

```
1  SELECT ?x WHERE { ?x rdf:type ap:Computer}

2  queryDataproperty(p);
3  queryObjectproperty(p, "providesPerceptionModule");
4  queryObjectproperty(p, "hasPowerConsumption");
5  queryObjectproperty(p, "providesCapability");
```

Listing 18: SPARQL Abfrage für „Computer“

- Cooling

```
1  SELECT ?x WHERE { ?x rdf:type ap:Cooling}

2  queryDataproperty(p);
3  queryObjectproperty(p, "providesCapability");
```

Listing 19 SPARQL Abfrage für „Cooling“

- Datalink

```
1  SELECT ?x WHERE { ?x rdf:type ap:Datalink}

2  queryDataproperty(p);
3  queryObjectproperty(p, "providesCapability");
```

Listing 20: SPARQL Abfrage für „Datalink“

- Mounting

```
1  SELECT ?x WHERE { ?x rdf:type ap:Mounting}

2  queryDataproperty(p);
3  queryObjectproperty(p, "canIntegrateSensor");
4  queryObjectproperty(p, "providesCapability");
```

Listing 21: SPARQL Abfrage für „Mounting“

- Powersupply

```
1  SELECT ?x WHERE { ?x rdf:type ap:Powersupply}

2  queryDataproperty(p);
3  queryObjectproperty(p, "providesCapability");
```

Listing 22: SPARQL Abfrage für „Powersupply“

- Sensor

```
1  SELECT ?x WHERE { ?x rdf:type ap:Sensor}
2
3  queryDataproperty(p);
4  queryObjectproperty(p, "requiresDaytimeCondition");
5  queryObjectproperty(p, "requiresWeatherCondition");
6  queryObjectproperty(p, "requiresCooling");
7  queryObjectproperty(p, "providesCapability");
```

Listing 23: SPARQL Abfrage für „Sensor“

B. Datenbank Abfragen für 7.3.2.3:

- *Alle Konfigurationen*: Es wurden alle möglichen UAV-Konfigurationen aus der Ontologie abgeleitet (vgl. Listing 15).
- *Individuum abfragen*: Es wurden alle Informationen für ein Individuum („*Sagitta-1*“) abgefragt.

```
1  SELECT ?p ?v  WHERE {  <ap:Sagitta-1> ?p ?v . ?p a owl:DatatypeProperty }
2  SELECT ?p ?v  WHERE {  <ap:Sagitta-1> ?p ?v . ?p a owl:ObjectProperty }
```

Listing 24: Abfrage für das Individuum "Sagitta-1"

- *Alle Individuen einer Klasse*: Es wurden alle Individuen der Klasse „*Platform*“ Klasse abgefragt

```
1  SELECT ?x WHERE { ?x rdf:type ap:Platform}
```

Listing 25: Abfrage für alle Instanzen der Klasse Platform

- *Individuum hinzufügen*: Es wurde ein Individuum („Sagitta-2“) in die Ontologie geschrieben.

```
1  try (RepositoryConnection conn = aRepo.getConnection()) {
2                 IRI iName = vf.createIRI(indi.getPrefix()+indi.getName());
3
4  // add Classes
5  for (String c : indi.getClasses()) {
6    IRI cName = vf.createIRI(indi.getPrefix()+c);
7    conn.add(iName, RDF.TYPE, cName);
8  }
9
10 //add dataproperties
11 for(Map.Entry<String, String> dp : indi.getDataproperties()){
12 IRI dataproperty = vf.createIRI(indi.getPrefix()+dp.getKey());
13
14 for(Map.Entry<String, String> e : dataproperties){
15 String tmpDP = indi.getPrefix()+dp.getKey();
16 if(tmpDP.equals(e.getKey())){
17 if(e.getValue().equals("http://...#integer")){
18 Literal value = vf.createLiteral(Integer.parseInt(dp.getValue()));
19 conn.add(iName, dataproperty, value);
20 }else if(e.getValue().equals("http://...#string")){
21 Literal value = vf.createLiteral(dp.getValue());
22 conn.add(iName, dataproperty, value);
23 }else if(e.getValue().equals("http://...#double")){
24 Literal value = vf.createLiteral(Double.parseDouble(dp.getValue()));
25 conn.add(iName, dataproperty, value);
26 }else if(e.getValue().equals("http://...#boolean")){
27 Literal value = vf.createLiteral(Boolean.parseBoolean(dp.getValue()));
28 conn.add(iName, dataproperty, value);
29 }else if(e.getValue().equals("http://...#Literal")){
30 Literal value = vf.createLiteral(dp.getValue());
31        conn.add(iName, dataproperty, value);
32 }else{
33    return false;
34 }}}}
35
36 //add objectproperties
37 for(Map.Entry<String, String> dp : indi.getObjectproperties()){
38        IRI objectproperty = vf.createIRI(indi.getPrefix()+dp.getKey());
39        IRI value = vf.createIRI(indi.getPrefix()+dp.getValue());
40        conn.add(iName, objectproperty, value);
41 }
42 return true;
43 }
```

Listing 26: Individuum "Sagitta-2" der Datenbank hinzufügen